IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
REQUEST FOR FILING APPLICATION UNDER 37 CFR 1.53(b)
WITHOUT FILING FEE OR EXECUTED INVENTOR'S DECLARATION

Assistant Commissioner for Patents
Washington, DC 20231

Atty. Dkt. 723-964

Date: November 28, 2000

Sir:

This is a request for filing a new PATENT APPLICATION under Rule 53(b) entitled:
METHOD AND APPARATUS FOR ANTI-ALIASING IN A GRAPHICS SYSTEM
without a filing fee and/or without an executed inventor's oath/declaration.
This application is made by the below identified inventor(s). Attached hereto are the following papers:

☒ An abstract together with
62 pages of specification and claims including
49 numbered claims and also attached is/are
18 sheets of accompanying drawings.
☐ This application is based on the following prior foreign application(s):

Application No.	Country	Filing Date

respectively, the entire content of which is hereby incorporated by reference in this application, and priority is hereby claimed therefrom.

☒ This application is based on the following prior provisional application(s):

Application No.	Filing Date
60/226,900	August 23, 2000

respectively, the entire content of which is hereby incorporated by reference in this application, and priority is hereby claimed therefrom.

☐ Certified copy/ies of foreign applications attached.
☐ This application is a ☐ continuation/☐ division/☐ continuation-in-part of application Serial No.   , filed   .
☐ Please amend the specification by inserting before the first line: --This application is a ☐ continuation/☐ division/ ☐ continuation-in-part of application Serial No.   , filed   , the entire content of which is hereby incorporated by reference in this application.--
☐ Please amend the specification by inserting before the first line: --This is a continuation of PCT application No.   , filed   , the entire content of which is hereby incorporated by reference in this application.--
☐ Please amend the specification by inserting before the first line: --This application claims the benefit of U.S. Provisional Application No., filed, the entire content of which is hereby incorporated by reference in this application.--

☐ Preliminary amendment to claims (attached hereto), to be entered before calculation of the fee.
☐ Also attached.

1. Inventor: Mark (first) M. MI Leather (last) USA (citizenship)
   Residence: (city) Saratoga   (state/country) CA
   Mailing Address: 12187 Woodside Dr., Saratoga, CA
   (Zip Code) 95070

2. Inventor: Farhad (first) MI FOULADI (last) USA (citizenship)
   Residence: (city) Palo Alto   (state/country) CA
   Mailing Address: 101 Alma St., Palo Alto, CA
   (Zip Code) 94031

☐ *See attached sheet(s) for additional inventor(s) information!!*

Address all future communications to NIXON & VANDERHYE P.C., 1100 North Glebe Road, 8th Floor, Arlington, Virginia 22201.

1100 North Glebe Road, 8th Floor
Arlington, Virginia 22201-4714
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JSP:mg

NIXON & VANDERHYE P.C.
By Atty: Joseph S. Presta, Reg. No. 35,329

Signature: _____

477558

U.S. PATENT APPLICATION

Inventor(s):   MARK M. LEATHER

                   FARHAD FOULADI

Invention: METHOD AND APPARATUS FOR ANTI-ALIASING IN A GRAPHICS

                     SYSTEM

NIXON & VANDERHYE P.C.
ATTORNEYS AT LAW
1100 NORTH GLEBE ROAD
8TH FLOOR
ARLINGTON, VIRGINIA 22201-4714
(703) 816-4000
Facsimile (703) 816-4100

SPECIFICATION

449223

Specification

METHOD AND APPARATUS FOR ANTI-ALIASING IN A GRAPHICS SYSTEM

Cross-Reference to Related Applications

5   This application is filed in accordance with 35 U.S.C. §119(e)(1) and claims the benefit of the provisional application Serial No. 60/226,900 filed on August 23, 2000, entitled "Method And Apparatus For Anti-Aliasing In A Graphics System."

This application is related to the following co-pending applications identified below (by title and attorney docket number), which focus on various 10 aspects of the graphics system described herein. Each of the following applications are hereby incorporated herein by reference.

- provisional Application No. 60/161,915, filed October 28, 1999 and its

   corresponding utility Application No. 09/465,754, filed December 17, 1999,

   both entitled "Vertex Cache For 3D Computer Graphics",

15 • provisional Application No. 60/226,912, filed August 23, 2000 and its

   corresponding utility Application No. _____, filed _____ (atty. dkt. no.

   723-959), both entitled "Method and Apparatus for Buffering Graphics Data in

   a Graphics System ",

- provisional Application No. 60/226,889, filed August 23, 2000 and its

20   corresponding utility Application No. _____, filed _____ (atty. dkt. no.

   723-958), both entitled "Graphics Pipeline Token Synchronization",

- provisional Application No. 60/226,891, filed August 23, 2000 and its corresponding utility Application No. _____ , filed _____ (atty. dkt. no. 723-961), both entitled "Method And Apparatus For Direct and Indirect Texture Processing In A Graphics System",

5
- provisional Application No. 60/226,888, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no. 723-968), both entitled "Recirculating Shade Tree Blender For A Graphics System",

- provisional Application No. 60/226,892, filed August 23, 2000 and its

10
corresponding utility Application No. _____, filed _____ (atty. dkt. no. 723-960), both entitled "Method And Apparatus For Efficient Generation Of Texture Coordinate Displacements For Implementing Emboss-Style Bump Mapping In A Graphics Rendering System",

- provisional Application No. 60/226,893, filed August 23, 2000 and its

15
corresponding utility Application No. _____ filed _____ (atty. dkt. no. 723-962), both entitled "Method And Apparatus For Environment-Mapped Bump-Mapping In A Graphics System",

- provisional Application No. 60/227,007, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no.

723-967), both entitled "Achromatic Lighting in a Graphics System and Method",

- provisional Application No. 60/226,910, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no.

5   723-957), both entitled "Graphics System With Embedded Frame Buffer Having Reconfigurable Pixel Formats",

- utility Application No. 09/585,329, filed June 2, 2000, entitled "Variable Bit Field Color Encoding" (atty. dkt. no. 723-749),

- provisional Application No. 60/226,890, filed August 23, 2000 and its

10   corresponding utility Application No. _____, filed _____ (atty. dkt. no. 723-956), both entitled "Method And Apparatus For Dynamically Reconfiguring The Order Of Hidden Surface Processing Based On Rendering Mode",

- provisional Application No. 60/226,915, filed August 23, 2000 and its

15   corresponding utility Application No. _____ filed _____ (atty. dkt. no. 723-973), both entitled "Method And Apparatus For Providing Non-Photorealistic Cartoon Outlining Within A Graphics System",

- provisional Application No. 60/227,032, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____, (atty. dkt. no.

723-954), both entitled "Method And Apparatus For Providing Improved Fog

Effects In A Graphics System",

- provisional Application No. 60/226,885, filed August 23, 2000 and its

corresponding utility Application No. _____, filed _____, (atty. dkt. no.

5   723-969), both entitled "Controller Interface For A Graphics System",

- provisional Application No. 60/227,033, filed August 23, 2000 and its

corresponding utility Application No. _____, filed _____ (atty. dkt. no.

723-955), both entitled "Method And Apparatus For Texture Tiling In A

Graphics System",

10 - provisional Application No. 60/226,899, filed August 23, 2000 and its

corresponding utility Application No. _____, filed _____ (atty. dkt. no.

723-971), both entitled "Method And Apparatus For Pre-Caching Data In

Audio Memory",

- provisional Application No. 60/226,913, filed August 23, 2000 and its

15   corresponding utility Application No. _____, filed _____ (atty. dkt. no.

723-965), both entitled "Z-Texturing",

- provisional Application No. 60/227,031, filed August 23, 2000 entitled

"Application Program Interface for a Graphics System" (atty. dkt. no. 723-880),

- provisional Application No. 60/227,030, filed August 23, 2000 and its

20   corresponding utility Application No. _____ , filed _____ (atty. dkt. no.

723-963), both entitled "Graphics System With Copy Out Conversions Between Embedded Frame Buffer And Main Memory",

- provisional Application No. 60/226,886, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no.

5   723-970), both entitled "Method and Apparatus for Accessing Shared Resources",

- provisional Application No. 60/226,884, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no. 723-972), both entitled "External Interfaces For A 3D Graphics and Audio

10   Coprocessor",

- provisional Application No. 60/226,894, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no. 723-974), both entitled "Graphics Processing System With Enhanced Memory Controller",

15 - provisional Application No. 60/226,914, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____, (atty. dkt. no. 723-966), both entitled " Low Cost Graphics System With Stitching Hardware Support For Skeletal Animation", and

- provisional Application No. 60/227,006, filed August 23, 2000 and its corresponding utility Application No. _____, filed _____ (atty. dkt. no. 723-953), both entitled " Shadow Mapping In A Low Cost Graphics System".

Field of the Invention

5   The present invention relates to computer graphics, and more particularly to interactive graphics systems such as home video game platforms. In more detail, the invention relates to anti-aliasing techniques for eliminating jagged edges from a computer graphics display. Still more particularly this invention relates to an

10   improved method and apparatus for full-scene anti-aliasing and de-flickering in a graphics system.

Background And Summary Of The Invention

Many of us have seen films containing remarkably realistic dinosaurs, aliens, animated toys and other fanciful creatures. Such animations are made

15   possible by computer graphics. Using such techniques, a computer graphics artist can specify how each object should look and how it should change in appearance over time, and a computer then models the objects and displays them on a display such as your television or a computer screen. The computer takes care of performing the many tasks required to make sure that each part of the displayed

20   image is colored and shaped just right based on the position and orientation of each object in a scene, the direction in which light seems to strike each object, the surface texture of each object, and other factors.

Because computer graphics generation is complex, computer-generated three-dimensional graphics just a few years ago were mostly limited to expensive specialized flight simulators, high-end graphics workstations and supercomputers. The public saw some of the images generated by these computer systems in movies and expensive television advertisements, but most of us couldn't actually interact with the computers doing the graphics generation. All this has changed with the availability of relatively inexpensive 3D graphics platforms such as, for example, the Nintendo 64® and various 3D graphics cards now available for personal computers. It is now possible to interact with exciting 3D animations and simulations on relatively inexpensive computer graphics systems in your home or office.

A problem graphics system designers confronted in the past is how to avoid bad visual effects associated with aliasing in a displayed image. Most modern computer graphics display devices create images by displaying an array of colored dots called pixels. Home color television sets and computer monitors work this way. When displaying graphics images on this kind of pixelated display, a staircasing effect can result due to the inherent characteristics of the graphics system and the display. Because the displayed digital image is made up of an array or grid of tiny pixels, edges of objects in the image may look jagged or stepped. For example, a smooth edge may appear as a stepped or jagged line due to the pixel grid. People refer to this stepped or jagged edge effect as "the jaggies" or "staircasing" -- but its technical name is "aliasing".

Aliasing is an inherent feature of a sampling based system. An unpleasant image can result when jaggies exist along edges and intersections of rendered primitives. Moreover, other visually disturbing side-effects of aliasing such as texture "swimming" or "flickering" can result throughout the entire rendered scene. These annoying side-effects are most often noticeable during animation.

Much work has been done in the past to solve the aliasing problem. More expensive graphics systems use "anti-aliasing" techniques that reduce or eliminate the visual effects of aliasing. One common anti-aliasing technique is based on a super-sampling/postfiltering approach. Using this approach, the graphics system

5   develops a sampled image that has more samples (sub-pixels) than the display device is capable of displaying. The graphics system filters the higher-resolution sampled image and resamples the image at the resolution of the display device. In simple terms, the graphics system intentionally coarsens the resolution of the sampled image before displaying it on the display device. In one example, the

10   graphics system might generate a certain number of sub-pixels for a pixel to be displayed on the screen, and blend the sub-pixels together to create the corresponding screen pixel.

Such anti-aliasing techniques improve the appearance of the image by reducing jaggies. The blurring or blending of pixels smoothes out edges -- even

15   though the image is still made up of discrete pixels -- because it provides a more gradual change in the pixel color pattern. As a result, the eye of the viewer perceives the edge as being much smoother and more accurate as compared to an aliased edge. It is not exactly intuitive that blurring could make the edge appear to be more accurate and realistic, but this is exactly how commonly used anti-aliasing

20   techniques work.

Unfortunately, however, the super-sampling anti-aliasing approach described above requires a substantial amount of memory and other resources. For example, storing n sub-pixels for each screen pixel requires a memory that is n times the size of what would otherwise be required. In addition, generating n sub-

25   pixels for each screen pixel requires the graphics pipeline to do a lot of extra work. Also, the blending operation can be very burdensome and can require additional circuitry or other processing resources. Consequently, such super-sampling

approaches to anti-aliasing have typically been found in the past in expensive graphics systems such as high end workstations but have been too "expensive" (in terms of required processing and memory resources) for use in low cost systems such as video game platforms.

5   Another anti-aliasing technique that has been used in the past involved the use of coverage values to reduce computational complexity and memory requirements. Such technique is described in U.S. Patent No. 5,742,277. This technique provides a method of anti-aliasing a silhouette edge by retrieving a color value for a silhouette edge pixel which falls on the silhouette edge from a frame
10   buffer, the retrieved color value representing a color of one or more foreground polygons which fall within the silhouette edge pixel. The technique estimates a background color of the silhouette edge pixel based on colors of neighboring pixels that are proximate to the silhouette edge pixel. This estimated background color represents a color of a portion of the silhouette edge pixel which is not occupied by
15   the one or more foreground polygons. An output color of the silhouette edge pixel is determined by interpolating between the retrieved color and the estimated background color. While this anti-aliasing technique can reduce jaggies in the rendered scene, the required estimation step has distinct disadvantages in terms of accuracy.

20   Another antialiasing approach is disclosed in US Patent Nos. 6,072,500 & 5,684,939. In this prior approach, a method for generating antialiased display data comprises storing a pixel memory that indicates a current state of a pixel that comprises a plurality of supersamples, wherein said pixel memory comprises a region mask having a plurality of fields, each field being associated with a unique
25   one of said supersamples; receiving a pixel packet, wherein said pixel packet indicates polygon coverage within said pixel, and a first color value; storing a second color value in an image memory, wherein said second color value is a

function of said first color value; determining a new pixel state based on said current pixel state and said pixel packet; updating said pixel memory based on said new pixel state, wherein if said new pixel state is a state in which the color value of each supersample is either said second color value or a third color value, 5 each of the fields associated with a supersample having said second color value stores an identifier that identifies said image memory; and generating antialiased display data based on said pixel memory. One drawback with this technique is that it requires region masks to be stored in pixel memory -- with a corresponding increase in the size and cost of the pixel memory.

In summary, although various full-scene anti-aliasing (FSAA) techniques 10 have been developed to mitigate the aliasing problem with varying degrees of success, some of the more effective approaches (for example, those involving conventional super-sampling and per-pixel object-precision area sampling), are often too computationally intensive and expensive to implement within a low cost 15 graphics system such as a home video game platform. Other techniques developed for lower cost systems have been partially effective, but suffer from accuracy problems. Therefore, while significant work has been done in the past, further improvements in anti-aliasing are desirable.

The present invention solves this problem by providing improved techniques 20 and arrangements for anti-aliasing in a graphics system.

In accordance with one aspect of our invention, we have developed particular techniques for anti-aliasing using an embedded frame buffer. For example, we have discovered particularly advantageous ways to perform anti-aliasing on the fly during a "copy out" process wherein an image data 25 representation is being transferred from an embedded frame buffer to another destination. Such techniques provide a highly efficient and cost-effective antialiasing approach that can be practically implemented in a low cost system.

We have also discovered ways to achieve higher anti-aliasing quality using a smaller number of multisamples than were formerly required. Typical existing multisample methods use "n" samples and a 1x1 box filter for reconstruction. We have discovered that by using a combination of particular sample patterns (i.e.,

5   multisample spatial distribution) and particular filter configurations that "share" some multisamples among several pixels, we can achieve better antialiasing than an "n" sample pattern and a reconstruction/antialiasing filter that extends across only single pixel area.

For example, using three multisamples per pixel and a 1x2 reconstruction

10   filter (i.e., a vertical filter that extends into one-half of the neighboring pixel areas immediately above and below the current pixel), and by using a specific sample pattern, we are able to achieve the equivalent of 6-sample antialiasing on vertical edges. Similarly, using a 1.33x2 reconstruction filter and a different sampling pattern, we achieve the equivalent of 6-sample antialiasing on vertical edges and 4-

15   sample antialiasing on horizontal edges. On a more intuitive level, we are intentionally varying (jittering) the sample pattern between pixels so as to achieve better antialiasing at the expense of noise along the edges; and then increasing the extent of the reconstruction filter to greater than 1x1 to reduce or eliminate the additional noise while sharing some multisamples between pixels for antialiasing

20   purposes -- thus achieving the effect of more multisamples than we are actually storing on a per-pixel basis in the frame buffer. This dramatic increase in antialiasing quality without requiring a corresponding increase in the number of multisamples stored in the frame buffer has particular advantages for low-cost graphics systems such as home video game platforms and personal computer

25   graphics cards.

Thus, in accordance with one aspect of the invention, a graphics system including graphics circuits coupled to an embedded frame buffer renders a

multisampled data representation of an image and stores the rendered multisampled data representation in the embedded frame buffer. We then resample said embedded frame buffer contents to provide an anti-aliased image. We can perform such antialiasing filtering on the image in the process of transferring the image from the embedded frame buffer to another location.

In accordance with yet another aspect of the invention, an anti-aliasing method implemented within a graphics system of the type that generates an image comprising plural pixels involves generating a multisampled data representation of an image having plural samples associated with each of the plural pixels. We resample the multisampled data representation to create an antialiased image for display. The resampling includes blending at least one of the plural samples into plural image pixels (i.e., sharing some of the multisamples between plural reconstructed screen pixels).

In accordance with yet another aspect provided by the invention, an anti-aliasing method comprises providing plural supersamples within each pixel of a pixel array. We vary the spatial distribution of the supersamples within neighboring pixels of the pixel array, and apply, to the array, an anti-aliasing filter having a pixel aperture including supersamples of at least two neighboring pixels.

In accordance with a more detailed aspect provided by the invention, an anti-aliasing method comprises:

- defining, within an embedded frame buffer, plural (e.g., three) super-sampled locations within each pixel of a pixel array, each said super-sampled location having a corresponding color value; and

- applying a vertical color data blending filter that blends a set of the pixel super-sampled color values during an operation that copies the embedded frame buffer out to an external destination.

By way of further non-limiting example, the following are some of the additional features provided by aspects of the invention:

- coverage masking of programmable super-sample locations efficiently generate a super-sampled image;

- a one-dimensional (e.g., vertical) filter applied during a copy-out operation from an embedded frame buffer to an external frame buffer can be used to blend the super-sampled image;

- super-samples from neighboring pixels can be included in the anti-aliased blend; and

- programmable locations and filtering weight(s) of the super-samples in the blend.

Another aspect of the invention provides, in a graphics system, a pixel data processing arrangement for providing full-scene anti-aliasing and/or de-flickering interlaced displays, comprising:

- a frame buffer containing super-sampled pixel data for a plurality of pixels;

- a plurality of scan-line buffers connected to receive super-sampled pixel color data from the frame buffer; and

- a multi-tap selectable-weight blending filter coupled to the scan-line buffers, the blending filter characterized by a vertically-arranged multiple-pixel filter support region wherein one or more color data samples from a plurality of vertically disposed pixels are blended to form a pixel color.

A further example anti-aliasing arrangement provided in accordance with an aspect of the invention includes:

- at least one storage location that defines plural super-sample locations within at least one pixel of a pixel array, each super-sample location having a corresponding color value;

- a coverage mask that specifies, for each of the plural super-sample locations within the at least one pixel, whether the plural super-sample locations are covered by rendered primitive fragments; and

- a one-dimensional color data blending filter that blends a resulting set of super-sample color values based on a programmable weighting function.

In one particular, non-limiting arrangement, the storage location may define three super-sample locations within the pixel. The filter may blend super-sample color values corresponding to the pixel with super-sample color values corresponding to at least one further pixel neighboring the pixel. The filter may blend super-sample color values corresponding to three vertically aligned pixels to produce a screen pixel output.

A further particular anti-aliasing technique provided in accordance with an aspect of the invention operates by:

- defining three sample locations for obtaining super-sampled color data associated with a pixel for each of a plurality of neighboring pixels;

- using a coverage mask to enable/disable samples corresponding to such locations, the coverage mask being based at least in part on corresponding portions of each pixel that are occupied by rendered primitive fragments; and

- blending resulting color data obtained from the locations to provide a pixel final color value.

A further aspect of the invention provides, for a pixel quad having first, second, third and fourth pixels and a quad center, a method of defining an optimal set of three super-sampling locations for anti-aliasing comprising:

- defining a first set of super-sample locations for a first pixel in the pixel quad at the following coordinates (range 1-12) relative to the quad center: (12,11) (4,7) (8,3);

- defining a second set of super-sample locations for a second pixel in the pixel quad at the following coordinates (range 1-12) relative to the quad center: (3,11) (11,7) (7,3);

- defining a third set of super-sample locations for a third pixel in the pixel quad at the following coordinates (range 1-12) relative to the quad center: (2,2) (10,6) (6,10); and

- defining a fourth set of super-sample locations for a fourth pixel in the pixel quad at the following coordinates (range 1-12) relative to the quad center: (9,2) (1,6) (5,6).

In still more detail, a preferred embodiment of the present invention provides efficient full-scene anti-aliasing by, *inter alia*, implementing a programmable-location super-sampling arrangement and using a selectable-weight vertical-pixel support area blending filter. For a 2 X 2 pixel group (quad), the locations of three samples within each super-sampled pixel are individually selectable. Preferably, a twelve-bit multi-sample coverage mask is used to determine which of twelve samples within a pixel quad are enabled based on the portions of each pixel occupied by a primitive fragment and any pre-computed z-buffering. Each super-sampled pixel is filtered during a copy-out operation from a local memory to an external frame buffer using a pixel blending filter arrangement that combines seven samples from three vertically arranged pixels. Three samples are taken from the current pixel, two samples are taken from a pixel immediately

above the current pixel and two samples are taken from a pixel immediately below the current pixel. A weighted average is then computed based on the enabled samples to determine the final color for the pixel. The weight coefficients used in the blending filter are also individually programmable. De-flickering of thin one-

5 pixel tall horizontal lines for interlaced video displays can be accomplished by using the pixel blending filter to blend color samples from pixels in alternate scan lines.

Brief Description Of The Drawings

These and other features and advantages provided by the invention will be

10 better and more completely understood by referring to the following detailed description of presently preferred embodiments in conjunction with the drawings, of which:

Figure 1 is an overall view of an example interactive computer graphics system;

15 Figure 2 is a block diagram of the Figure 1 example computer graphics system;

Figure 3 is a block diagram of the example graphics and audio processor shown in Figure 2;

Figure 4 is a block diagram of the example 3D graphics processor shown in

20 Figure 3;

Figure 5 is an example logical flow diagram of the Figure 4 graphics and audio processor;

Figure 6 shows an example anti-aliasing process;

Figure 6A shows an exemplary flowchart of the anti-aliasing method of the

25 instant invention;

Figure 7 shows an exemplary primitive and super-sampled pixel quad;

Figure 8 shows an exemplary sampling pattern for a quad;

Figure 9 shows how each pixel is divided into a 12x12 subpixel grid where super-sample locations can be defined;

Figure 10 shows a preferred super sample patterns for particular corresponding reconstruction filter configuration;

Figure 11 shows exemplary control registers for setting sample location and filter coefficients;

Figure 12 shows an exemplary super-sampling coverage mask for a pixel quad of the type shown in Figure 7;

Figure 13 shows an exemplary copy-out pipeline for the graphics processor of Figure 3 between the embedded frame buffer and the external frame buffer;

Figure 14 shows a vertical filter blending programmable 7-tap filter used for anti-aliasing in accordance with the instant invention;

Figure 15 shows an example vertical filter aperture;

Figure 16 shows an example vertical filter structure;

Figure 17 shows an example anti-aliasing copy out buffering operation;

Figure 18 shows a block diagram of the anti-aliasing buffering used in accordance with a preferred embodiment of the instant invention;

Figure 19 shows the filter of Figure 14 used in a non-anti-aliasing mode and which reduces flickering in accordance with the instant invention;

Figure 20 show a block diagram of the de-flickering buffering used in accordance with a preferred embodiment of the instant invention; and

Figures 21A and 21B show example alternative compatible implementations

Detailed Description Of Example Embodiments Of The Invention

Figure 1 shows an example interactive 3D computer graphics system 50. System 50 can be used to play interactive 3D video games with interesting stereo sound. It can also be used for a variety of other applications.

5   In this example, system 50 is capable of processing, interactively in real time, a digital representation or model of a three-dimensional world. System 50 can display some or all of the world from any arbitrary viewpoint. For example, system 50 can interactively change the viewpoint in response to real time inputs from handheld controllers 52a, 52b or other input devices. This allows the game

10   player to see the world through the eyes of someone within or outside of the world. System 50 can be used for applications that do not require real time 3D interactive display (e.g., 2D display generation and/or non-interactive display), but the capability of displaying quality 3D images very quickly can be used to create very realistic and exciting game play or other graphical interactions.

15   To play a video game or other application using system 50, the user first connects a main unit 54 to his or her color television set 56 or other display device by connecting a cable 58 between the two. Main unit 54 produces both video signals and audio signals for controlling color television set 56. The video signals are what controls the images displayed on the television screen 59, and the audio

20   signals are played back as sound through television stereo loudspeakers 61L, 61R.

The user also needs to connect main unit 54 to a power source. This power source may be a conventional AC adapter (not shown) that plugs into a standard home electrical wall socket and converts the house current into a lower DC voltage signal suitable for powering the main unit 54. Batteries could be used in other

25   implementations.

The user may use hand controllers 52a, 52b to control main unit 54. Controls 60 can be used, for example, to specify the direction (up or down, left or right, closer or further away) that a character displayed on television 56 should move within a 3D world. Controls 60 also provide input for other applications

5 (e.g., menu selection, pointer/cursor control, etc.). Controllers 52 can take a variety of forms. In this example, controllers 52 shown each include controls 60 such as joysticks, push buttons and/or directional switches. Controllers 52 may be connected to main unit 54 by cables or wirelessly via electromagnetic (e.g., radio or infrared) waves.

10 To play an application such as a game, the user selects an appropriate storage medium 62 storing the video game or other application he or she wants to play, and inserts that storage medium into a slot 64 in main unit 54. Storage medium 62 may, for example, be a specially encoded and/or encrypted optical and/or magnetic disk. The user may operate a power switch 66 to turn on main

15 unit 54 and cause the main unit to begin running the video game or other application based on the software stored in the storage medium 62. The user may operate controllers 52 to provide inputs to main unit 54. For example, operating a control 60 may cause the game or other application to start. Moving other controls 60 can cause animated characters to move in different directions or change the

20 user's point of view in a 3D world. Depending upon the particular software stored within the storage medium 62, the various controls 60 on the controller 52 can perform different functions at different times.

Example Electronics of Overall System

Figure 2 shows a block diagram of example components of system 50. The

25 primary components include:

- a main processor (CPU) 110,

- a main memory 112, and

- a graphics and audio processor 114.

In this example, main processor 110 (e.g., an enhanced IBM Power PC 750) receives inputs from handheld controllers 108 (and/or other input devices) via

5   graphics and audio processor 114. Main processor 110 interactively responds to user inputs, and executes a video game or other program supplied, for example, by external storage media 62 via a mass storage access device 106 such as an optical disk drive. As one example, in the context of video game play, main processor 110 can perform collision detection and animation processing in addition to a variety of

10   interactive and control functions.

In this example, main processor 110 generates 3D graphics and audio commands and sends them to graphics and audio processor 114. The graphics and audio processor 114 processes these commands to generate interesting visual images on display 59 and interesting stereo sound on stereo loudspeakers 61R, 61L

15   or other suitable sound-generating devices.

Example system 50 includes a video encoder 120 that receives image signals from graphics and audio processor 114 and converts the image signals into analog and/or digital video signals suitable for display on a standard display device such as a computer monitor or home color television set 56. System 50 also includes an

20   audio codec (compressor/decompressor) 122 that compresses and decompresses digitized audio signals and may also convert between digital and analog audio signaling formats as needed. Audio codec 122 can receive audio inputs via a buffer 124 and provide them to graphics and audio processor 114 for processing (e.g., mixing with other audio signals the processor generates and/or receives via a

25   streaming audio output of mass storage access device 106). Graphics and audio processor 114 in this example can store audio related information in an audio memory 126 that is available for audio tasks. Graphics and audio processor 114

provides the resulting audio output signals to audio codec 122 for decompression and conversion to analog signals (e.g., via buffer amplifiers 128L, 128R) so they can be reproduced by loudspeakers 61L, 61R.

Graphics and audio processor 114 has the ability to communicate with various additional devices that may be present within system 50. For example, a parallel digital bus 130 may be used to communicate with mass storage access device 106 and/or other components. A serial peripheral bus 132 may communicate with a variety of peripheral or other devices including, for example:

- a programmable read-only memory and/or real time clock 134,

- a modem 136 or other networking interface (which may in turn connect system 50 to a telecommunications network 138 such as the Internet or other digital network from/to which program instructions and/or data can be downloaded or uploaded), and

- flash memory 140.

A further external serial bus 142 may be used to communicate with additional expansion memory 144 (e.g., a memory card) or other devices. Connectors may be used to connect various devices to busses 130, 132, 142.

Example Graphics And Audio Processor

Figure 3 is a block diagram of an example graphics and audio processor 114. Graphics and audio processor 114 in one example may be a single-chip ASIC (application specific integrated circuit). In this example, graphics and audio processor 114 includes:

- a processor interface 150,

- a memory interface/controller 152,

- a 3D graphics processor 154,

- an audio digital signal processor (DSP) 156,

- an audio memory interface 158,

- an audio interface and mixer 160,

- a peripheral controller 162, and

5   • a display controller 164.

3D graphics processor 154 performs graphics processing tasks. Audio digital signal processor 156 performs audio processing tasks. Display controller 164 accesses image information from main memory 112 and provides it to video encoder 120 for display on display device 56. Audio interface and mixer 160

10   interfaces with audio codec 122, and can also mix audio from different sources (e.g., streaming audio from mass storage access device 106, the output of audio DSP 156, and external audio input received via audio codec 122). Processor interface 150 provides a data and control interface between main processor 110 and graphics and audio processor 114.

15   Memory interface 152 provides a data and control interface between graphics and audio processor 114 and memory 112. In this example, main processor 110 accesses main memory 112 via processor interface 150 and memory interface 152 that are part of graphics and audio processor 114. Peripheral controller 162 provides a data and control interface between graphics and audio

20   processor 114 and the various peripherals mentioned above. Audio memory interface 158 provides an interface with audio memory 126.

Example Graphics Pipeline

Figure 4 shows a more detailed view of an example 3D graphics processor 154. 3D graphics processor 154 includes, among other things, a command

25   processor 200 and a 3D graphics pipeline 180. Main processor 110 communicates

streams of data (e.g., graphics command streams and display lists) to command processor 200. Main processor 110 has a two-level cache 115 to minimize memory latency, and also has a write-gathering buffer 111 for uncached data streams targeted for the graphics and audio processor 114. The write-gathering

5   buffer 111 collects partial cache lines into full cache lines and sends the data out to the graphics and audio processor 114 one cache line at a time for maximum bus usage.

Command processor 200 receives display commands from main processor 110 and parses them -- obtaining any additional data necessary to process them

10   from shared memory 112. The command processor 200 provides a stream of vertex commands to graphics pipeline 180 for 2D and/or 3D processing and rendering. Graphics pipeline 180 generates images based on these commands. The resulting image information may be transferred to main memory 112 for access by display controller/video interface unit 164 -- which displays the frame

15   buffer output of pipeline 180 on display 56.

Figure 5 is a logical flow diagram of graphics processor 154. Main processor 110 may store graphics command streams 210, display lists 212 and vertex arrays 214 in main memory 112, and pass pointers to command processor 200 via bus interface 150. The main processor 110 stores graphics commands in

20   one or more graphics first-in-first-out (FIFO) buffers 210 it allocates in main memory 110. The command processor 200 fetches:

- command streams from main memory 112 via an on-chip FIFO memory buffer 216 that receives and buffers the graphics commands for synchronization/flow control and load balancing,

25   - display lists 212 from main memory 112 via an on-chip call FIFO memory buffer 218, and

- vertex attributes from the command stream and/or from vertex arrays 214 in main memory 112 via a vertex cache 220.

Command processor 200 performs command processing operations 200a that convert attribute types to floating point format, and pass the resulting complete vertex polygon data to graphics pipeline 180 for rendering/rasterization. A programmable memory arbitration circuitry 130 (see Figure 4) arbitrates access to shared main memory 112 between graphics pipeline 180, command processor 200 and display controller/video interface unit 164.

Figure 4 shows that graphics pipeline 180 may include:

- a transform unit 300,

- a setup/rasterizer 400,

- a texture unit 500,

- a texture environment unit 600, and

- a pixel engine 700.

Transform unit 300 performs a variety of 2D and 3D transform and other operations 300a (see Figure 5). Transform unit 300 may include one or more matrix memories 300b for storing matrices used in transformation processing 300a. Transform unit 300 transforms incoming geometry per vertex from object space to screen space; and transforms incoming texture coordinates and computes projective texture coordinates (300c). Transform unit 300 may also perform polygon clipping/culling 300d. Lighting processing 300e also performed by transform unit 300b provides per vertex lighting computations for up to eight independent lights in one example embodiment. Transform unit 300 can also perform texture coordinate generation (300c) for embossed type bump mapping effects, as well as polygon clipping/culling operations (300d).

Setup/rasterizer 400 includes a setup unit which receives vertex data from transform unit 300 and sends triangle setup information to one or more rasterizer units (400b) performing edge rasterization, texture coordinate rasterization and color rasterization.

5   Texture unit 500 (which may include an on-chip texture memory (TMEM) 502) performs various tasks related to texturing including for example:

- retrieving textures 504 from main memory 112,

- texture processing (500a) including, for example, multi-texture handling, post-cache texture decompression, texture filtering, embossing, shadows and lighting through the use of projective textures, and BLIT with alpha transparency and depth,

- bump map processing for computing texture coordinate displacements for bump mapping, pseudo texture and texture tiling effects (500b), and

- indirect texture processing (500c).

15   Texture unit 500 outputs filtered texture values to the texture environment unit 600 for texture environment processing (600a). Texture environment unit 600 blends polygon and texture color/alpha/depth, and can also perform texture fog processing (600b) to achieve inverse range based fog effects. Texture environment unit 600 can provide multiple stages to perform a variety of other interesting

20   environment-related functions based for example on color/alpha modulation, embossing, detail texturing, texture swapping, clamping, and depth blending..

Pixel engine 700 performs depth (z) compare (700a) and pixel blending (700b). In this example, pixel engine 700 stores data into an embedded (on-chip) frame buffer memory 702. Graphics pipeline 180 may include one or more

25   embedded DRAM memories 702 to store frame buffer and/or texture information locally. Z compares 700a' can also be performed at an earlier stage in the graphics pipeline 180 depending on the rendering mode currently in effect (e.g., z compares

can be performed earlier if alpha blending is not required). The pixel engine 700
includes a copy operation 700c that periodically writes on-chip frame buffer 702 to
main memory 112 for access by display/video interface unit 164. This copy
operation 700c can also be used to copy embedded frame buffer 702 contents to
5   textures in the main memory 112 for dynamic texture synthesis effects. Anti-
aliasing and other filtering can be performed during the copy-out operation from
the embedded frame buffer (EFB) to the external frame buffer (XFB). The frame
buffer output of graphics pipeline 180 (which is ultimately stored in main memory
112) is read each frame by display/video interface unit 164. Display
10   controller/video interface 164 provides digital RGB pixel values for display on
display 102.

Example Anti-aliasing Techniques and Arrangements

As shown in Figure 6, anti-aliasing is performed in two main phases in the
example embodiment. The first phase occurs during rendering and involves
15   rasterizing the image into a super-sampled embedded frame buffer (EFB) (block
550). The second phase performed during a copy-out operation involves
filtering/blending the super-samples to create screen pixel output colors (block
552). The copied-out image is then displayed on display 56 (block 554). Figure
6A provides a more detailed summary block diagram of the anti-aliasing in
20   accordance with an aspect of this instant invention. The anti-aliasing starts by
defining three multisample locations per pixel for a current 2x2 pixel quad (block
552). Programmable control registers are used to determine the location of the
samples in units of 1/12 pixel (block 550a-1). A multisample coverage mask is
then created for enabling/disabling samples based on the portion of the pixel
25   occupied by a primitive (block 550b). If early z-compare is enabled, the coverage
mask can also be affected by the primitive depth (block 550b). Once a frame is

completed and all primitives have been rendered into the embedded frame buffer, the color data from enabled samples is blended, using programmable weighting coefficients (block 552a), from three vertically aligned neighboring pixels during copy-out from the local memory (EFB) to main memory (XFB) (block 552). The

5   anti-aliased pixel color is then displayed from the XFB by the video interface unit (block 554).

Each of the two basic phases of anti-aliasing (rasterizing and filtering) will be described in detail below.

Rasterizing/Rendering Phase

10   The first anti-aliasing phase (Figure 6, block 550) occurs when the rasterizer (400b) is performing edge rasterization into the embedded frame buffer (EFB) 702. Preferably, this rasterizer is an edge and z rasterizer which generates x, y, z and coverage mask values for programmable super-sample locations within every visible pixel quad contained by the current triangle or other primitive . The

15   primitive information is preferably received from the setup unit 400 in a form suitable for easy rasterization. The process is repeated for each primitive in the image.

Pixel Quads Have Programmable Subpixel Locations

Figure 7 shows an exemplary pixel quad 610 with an example primitive

20   (e.g., triangle) 612 overlaid thereon. The pixel quad includes 4 pixels (Pix00, Pix01, Pix10 and Pix11) in a 2x2 configuration. Within each pixel (e.g. Pix00) in the quad 610, three super-sample locations (e.g. S0, S1 and S2) are programmably selected and specified. This results in twelve super-sample locations per pixel quad 610. While the graphics pipeline described above processes pixels in quads,

25   other arrangements are also possible.

Irrespective of how many pixels are generated in parallel, each pixel Pix in the example embodiment includes plural (e.g., three in one particular implementation) sub-pixels (see Figure 8). In the example arrangement, the location of each of these sub-pixels within the pixel quad is programmable.

5   Furthermore, in the example embodiment, each pixel quad within a pixel array comprising thousands of pixels includes twelve such sub-pixels at corresponding programmable locations. As can be seen most clearly in Figure 8, the three sample locations S within each pixel of the quad 610 can be different for each neighboring pixel. Figure 8 shows one example of three sample locations within each pixel. In

10   accordance with an aspect of the invention, we can vary ("jitter") the sample pattern between pixels (e.g., the spatial distribution of the multi-sample locations within the neighboring pixels) so as to achieve better antialiasing at the expense of noise along the edges. We can then increase the extent of the reconstruction filter to greater than 1x1 to reduce the noise. Increasing the reconstruction filter to

15   greater than 1x1 means that some multisamples are shared between different pixels, i.e., they contribute to the anti-aliased screen pixel output color of more than one screen pixel for display. Thus, we get the effect of more multisamples per pixel than we are actually storing in the frame buffer. In certain cases, with a carefully constructed supersample pattern, the additional filter is able to cancel out

20   the noise entirely.

The programmer can set the subsample locations by writing global registers. In this particular embodiment, super-sample locations may be specified as x and y distance (e.g., in units related to pixel size, e.g., 1/12 pixel), from the pixel quad center. In other arrangements (e.g., those not based on pixel quads), different

25   approaches to specifying the multisample locations within the various pixels can be used. Since the location of each of the super-samples in each pixel is programmable in the example embodiment in the example embodiment, the

particular sampling locations (S0-S1) for each quad can be changed as desired for the particular application. On the other hand, in alternative embodiments, a particularly optimal multisample location values could be fixed (e.g., set in hardware) so the application programmer does not need to worry about it. Thus, while the locations are programmable in the example embodiment, a hardwired optional pattern could be used in lieu of programmability. Whatever pattern is selected, it can be repeated across a certain number of neighboring pixels in a frame.

One convenient way to specify the particular spatial distribution of multisamples within the pixel array in the example embodiment is to specify multisample locations within a pixel quad relative to the center of the pixel quad. Figure 9 shows that in the example embodiment, each pixel in the quad is broken down into a 12x12 grid. Each sample has a specified x and y distance (in units of 1/12 pixel) from the center of the quad. Thus, xsij,ysij , where j=(0-2) and i=(0-3), specifies the location (x and y coordinate) of multisample location j in pixel i for the quad.

Figure 10 shows an enlarged view of a quad wherein preferred example sample locations are shown for each pixel in the quad. This preferred pattern has been determined to provide good results for certain applications. In particular, this preferred sampling pattern works well for a system that provides three multisamples within each pixel and a 1 x 2 reconstruction filter vertical filter coefficients (1/12, 1/6, 1/6, 1/6, 1/6, 1/6, 1/12) (see Figures 14 & 16) -- which filter will cancel out the noise intentionally introduced into the Figure 10 multisample distribution pattern through jittering of multisample locations within neighboring pixels. By "1x2" reconstruction filter, we mean that the filter aperture extends across a single pixel in the horizontal (x) dimension and extends across two pixels in the vertical (y) dimension (i.e., the filter "aperture" covers the entire area of the

current pixel whose output is being generated along with half of area of the pixel just above the current pixel and half of the area of the pixel just below the current pixel). Note that the Figure 10 pattern should be used with the appropriately configured reconstruction filter described above; if the Figure 10 pattern is used

5   without the corresponding filter, images will look worse than they would without any antialiasing.

Thus, in accordance with this aspect of the invention, there is a relationship between (a) the number and locations of multisamples, and (b) the aperture and weighting coefficients of the anti-aliasing (reconstruction) filter. The combination

10 of a specific sample pattern and a specific filter can give substantially better antialiasing than a "n" sample pattern and 1x1 filter alone for a given number of multisamples per pixel.

In Figure 10, and assuming that numbers 1-12 are used for the x and y

15 (distances) scale from the center of the quad, the preferred locations are as follows:

for PixelId 0: $xs00 = 12,$ $ys00 = 11$

$xs01 = 4,$ $ys01 = 7$

$xs02 = 8,$ $ys02 = 3$

for PixelId 1: $xs10 = 3,$ $ys10 = 11$

20 $xs11 = 11,$ $ys11 = 7$

$xs12 = 7,$ $ys12 = 3$

for PixelId 2: $xs20 = 2,$ $ys20 = 2$

$xs21 = 10,$ $ys21 = 6$

$xs22 = 6,$ $ys22 = 10$

25 for PixelId 3: $xs30 = 9,$ $ys30 = 2$

$xs31 = 1,$ $ys31 = 6$

$xs32 = 5,$ $ys32 = 10$

While the three sample points are numbered (0-2) in Figure 10 based on their relative position from the top of each pixel, the samples may be numbered based on their y distance from the center of the quad. The numbering is taken into account during the filtering operations described below, wherein the pixel having the greatest y value in each quad may not be used by the filter.

Other sample patterns may be used to allow us to use a different number of multisamples per pixel and/or a different filter aperture. In this particular implementation of our invention, we have chosen to use 3 samples per pixel and a 1x2 reconstruction filter. By using a specific sample pattern, we are able to achieve the equivalent of 6-sample antialiasing on vertical edges. However, in another example implementation, we can use a 1.33x2 reconstruction filter (i.e., a horizontal and vertical reconstruction filter with an aperture in the horizontal or x dimension that extends across the current pixel being generated and also covers 1/6 of the neighboring pixel immediately to the left of the current pixel and 1/6 of the area of the neighboring pixel immediately to the right of the neighboring pixel) with a different sampling pattern, and achieve the equivalent of 6-sample antialiasing on vertical edges, and 4-sample antialiasing on horizontal edges.

In another embodiment, another sample pattern may be used in a configuration having 4 samples per pixel and a 1.5 x 1.5 reconstruction filter. The sample pattern may achieve the equivalent of 6 sample antialiasing on both horizontal and vertical edges with only 4 supersamples stored in the frame buffer for each pixel. The filtering would be barely noticeable, however the increase in anti-aliasing quality would be significant. Subjective tests show a very big jump in perceived quality between 4 and 6 samples, and the quality curve is then fairly flat after 6 samples (e.g. most users cannot tell the difference between 6 samples and 16). This technique thus allows good quality antialiasing using 4 samples (as

opposed to today's existing lower quality antialiasing), and eliminates the need (and expense) of going to 8 multisamples per pixel to achieve good quality.

It is noted that the perceived visual quality of the antialiasing is generally subjective, in that it depends somewhat on the preferences of the particular

5   individual viewing the resulting image. However, we have found that advantageous patterns and corresponding filter dimensions can be used which provide an increased perceived visual quality for many viewers. The selection of a particular pattern is generally done using a trial and error procedure. However, exemplary general criteria that may be used when selecting a pattern are as

10   follows: 1) setting the pattern such that when moving a vertical edge horizontally one new sample is hit every, for example, $1/6^{th}$ of a pixel, thereby providing fairly even gray scale stepping; 2) setting the pattern such that when moving a horizontal edge vertically one new sample is hit every, for example, $1/6^{th}$ of a pixel, thereby also providing fairly even gray scale stepping; and 3) setting the

15   pattern such that the samples are spread out as much as possible so that no clustering is visible when the pattern is viewed from a distance.

Example Techniques for Programming Multisample Location/Spatial Distribution in Specific Disclosed Example Implementation

20   Figure 11 shows exemplary global registers (LOC0-LOC3) 616a, 616b, 616c and 616d for storing the x and y coordinate of each of the multisample locations within the quad 610 in the example detailed implementation. These locations can be programmed for each of the pixels in the quad using, for example, commands in the application program interface (API) for the graphics processor. These global

25   registers 616a, 616b, 616c and 616d are used by the pixel engine to specify the

sample locations. The following is an example API command or function which sets up these exemplary registers for this purpose:

GXSetAnti-aliasing:

Argument:

5

| GXBool | Mode; | //Enable Anti-aliasing mode |
| u8 | SamplePts[4][3][2] | //Location of multisample points |

This function sets anti-aliasing mode. The application also sets the appropriate pixel format for anti-aliasing. It is noted that this mode is not per-primitive, but per-frame. The SamplePts array specifies the location of multisample points per pixel quad. The point offsets are given in 1/12 pixel units. There are three points per pixel. The sample points may be specified as follows:

SamplePts [pixelId][pointId][x/y]

where,

PixelId 0 = upper left pixel in a 2x2 block

PixelId 1= upper right pixel

PixelId 2 = lower left pixel

PixelId 3 = lower right pixel

PointId [0-2] = one of the three multisample points for the pixel.

x/y   = x or y coordinate of the sample point.

Figure 11 also shows an exemplary global register (Mode) 618 which includes a bit (ms_en) specifying whether or not multisampling is enabled. The ms_en bit is enabled for anti-aliasing. Thus, the system preferably enables selective operation of the anti-aliasing mode. This register 618 may also be used to specify other parameters for the system. For example, the ntex bits can be used to specify the number of sets of texture coordinates passed from the transform unit to the setup unit, and the number of sets of texture coordinates passed from the setup

unit to the rasterization unit for texture coordinate rasterization. The ncol bits can be used to specify the number of sets of color values passed from the transform unit to the setup unit, and the number of sets of color values passed from the setup unit to the rasterizer unit for color rasterization. The reject_en bits can be used to

5   specify how to reject triangles based on whether they are front or back facing (i.e. reject none, front, back or all). The flat_en bit can be used to specify that triangles are flat shaded. The n_tev bits can be used to specify the number of texture environment (TEV) operations currently defined.

Coverage Masks

10   As shown in Figure 12, coverage masks 614 are generated for the pixel quads 610. The coverage mask 614 specifies which of the super-sample locations (S0-S2) are covered by each of the primitive fragments 612 being rasterized. In this context, the term "primitive fragment" refers to the portion of the current primitive being rendered that intersects with the pixel(s) currently being evaluated

15   by the edge rasterizer 400b. The edge rasterizer 400b determines coverage in the process of rasterizing the various edges of primitive fragments 612. The coverage mask 614 is set according to the edge equation data for the current primitive developed by the rasterizer 400b.

In the example embodiment, the coverage mask 614 includes 12 bits, i.e.

20   three bits for each pixel in the quad – each of the three bits corresponding to a different subpixel in the pixel. The coverage mask bits are set based on whether or not a primitive fragment is covering each of the respective super-sample locations in the quad 610.

The coverage mask 614 is used to enable/disable super-samples

25   corresponding to the programmed super-sample locations based on whether the super-sample is covered by the current primitive fragment. The coverage mask

records which of the super-sample locations of each pixel are occupied (covered) by a primitive fragment, and which of the super-sample locations are not occupied (uncovered) by the primitive fragment. This information is used in the example embodiment to realize efficiency improvements in the Z buffering operations to

5   follow. While coverage masks are helpful in generating the various multisamples in the embedded frame buffer 702, they are not essential; other techniques not relying on coverage masks could be used in other implementations.

Z Buffering

The disclosed embodiment embedded frame buffer 702 includes a z (depth)

10 buffer as well as a color buffer. The example embodiment uses the coverage mask 614 to determine whether or not to perform z buffering for each subsample location S. If a subsample location S is not covered by a primitive fragment, there is no need to perform a z compare for that subsample since the corresponding z and color values within frame buffer 702 will not be updated in any event. If the

15 subsample location S is covered by the primitive fragment, then the z compare operation 700a is performed for that subsample to determine whether the primitive fragment is hidden or visible at that subsample location -- and blend operation 700b conditionally blends colors into the embedded color frame buffer if the z compare 700a indicates that pixel fragment is visible at that subsample location.

20 To reduce the number of wires in the example implementation, a single 28-bit quad Z value with the format of 26.1 and Zx and Zy with format s26.5 are sent to the z compare 700a. The quad Z value is the value of pixel Z at the center of the pixel quad. Two 32-bit adders and two 5x32 bit multipliers can solve the plane equation for each subsample location by performing the following equation to

25 extrapolate for the twelve samples in a pixel quad using the quad Z value and slope information obtained from the rasterizer 400b:

$$Z(dx, dy) = Z + (Zx)(dx) + (Zy)(dy)$$

where dx and dy are based on the pixel number and super-sample location (clamping may be performed as well to prevent overflow.

As mentioned above, Z comparison 700a' (see Figure 5) can be performed at an earlier stage in the graphics pipeline 180 depending on the rendering mode currently in effect (e.g., Z compares can be performed if alpha test is not required). In such case, Z compare 700a' may conditionally perform Z buffering (depth compare and write) and on the extrapolated Z values developed by the plane equation discussed above. As a result of this operation, a number of bits and the coverage mask 614 may be cleared due to the Z buffering operation -- thus allowing the cover mask to carry the results of the z compare down to a later stage in the pipeline where updating of the color frame buffer occurs. The resulting coverage mask, along with the x,y location of the pixel quad, is passed to texture coordinate rasterization processes preliminary to further processing by texture block 500a. If the "Z before texture" function is not enable, then Z compare block 700a (see Figure 5) performs the Z compare and write prior to writing color into frame buffer 702 – and the coverage mask is not changed by the results of the Z compare (but may of course still be used to determine whether a Z compare is even necessary for a particular subsample location).

Example Copy Out and Vertical Filtering

Once all primitives of a scene have been processed and a super-sampled image has been rendered into the embedded frame buffer (EFB) in the manner described above, the second phase of anti-aliasing (see Figure 6A, block 552) can be performed as described below. However, before moving to the second phase, a brief explanation of the exemplary embedded frame buffer (EFB) is provided in order to provide a better understanding of the exemplary embodiment.

Example Embedded Frame Buffer Configuration

In this example, the embedded frame buffer (EFB) has a memory capacity of approximately 2MB. The maximum pixel width and height of the frame buffer is determined by the size of each pixel. In this example, there are several different

5   pixel sizes including for example:

- 48-bit color and z; and

- 96-bit super-sampled color and Z

The formats can preferably be set using the API. An example API function for this purpose is as follows:

10   GXSetPixelFormat

Argument:

GXPixelFormats	Format	//Sets pixel format for frame buffer
GXZCmprFormats	ZCmpr	//Sets compression format for 16 bit z
GXBool	Ztop	//Z compare before texture

15

This function sets the format of the frame buffer. The function is called before any drawing operations are performed. The pixel format cannot be changed in the middle of a frame in the example embodiment. The 16 bit Z values (in multisample anti-aliasing mode) can be uncompressed or compressed. The

20   compressed values give a better precision and range. The Ztop flag can be used to perform depth comparisons before texture mapping (Z-before-texture). This improves the texture bandwidth because less texels need to be fetched and filtered.

The 48-bit format for the embedded frame buffer (EFB) is intended for non-anti-aliasing, and has the following features:

25   • 24-bit color (either 8/8/8 with no alpha, or 6/6/6/6 with 6 bits of alpha)

- 24-bit Z.

In this mode, the format can support a maximum resolution of 640x528. The width must be between 0-640 and the EFB stride is fixed at 640 pixels.

The 96-bit super-sampling format is used for anti-aliasing and has the following features:

- 3 samples of 16-bit color (5 bits of Red, 6 bits of Green, 5 bits of Blue, no alpha)

- 3 samples of 16-bit Z (depth).

This format can support a maximum resolution of 640x264. The width is preferably between 0-640 and the stride is fixed at 640.

As can be seen from the above, while anti-aliasing increases visual quality on the polygon edges and intersections, it does cost performance and Z quality. Anti-aliasing uses the 96 bit super-sampling EFB format that requires twice as much memory as 48-bit point sampled pixels. This mode also reduces Z buffering precision to 16 bits rather than 24 bits in other formats. Anti-aliasing also reduces peak fill rate from 800Mpixels/s to 400Mpixels/s. However, if more that one stage is employed in the texture environment unit (TEV), this reduction is hidden, in that, in this example, using two TEV stages also reduces the fill rate to 400Mpixels/s.

In one embodiment, the rendering rate with anti-aliasing activated drops down to two pixels/clock due to the embedded frame buffer 702 bandwidth limitations. However, if two or more textures are turned on, the rate at which pixel quads are sent to the pixel engine 700 drops down to less than or equal to one pixel quad every two clocks in this particular embodiment. In this case, turning on anti-aliasing will not impact fill rate. Thus, if a particular scene is geometry-limited, then anti-aliasing will not adversely impact rendering performance. On the other hand, if a particular scene is fill-limited, rendering performance may be

substantially adversely impacted by activating anti-aliasing as opposed to using the point sampled mode. The same application can activate and deactivate anti-aliasing for different scenes and different images depending on whether the scenes or images are geometry-limited or fill-limited – or depending upon the image

5 quality required in a particular scene or image. The ability to dynamically activate and deactivate anti-aliasing on a frame-by-frame basis provides great flexibility in allowing an application programmer to make tradeoffs between image quality and speed performance.

Example Copy Out Operation

10 The second stage of anti-aliasing occurs during copy-out from the embedded frame buffer (EFB) 702 to the display buffer 113 (external frame buffer (XFB)) in main memory 112. An example copy-out pipeline 620 for this example is shown in Figure 13.

As shown in Figure 13, the copy-out pipeline 620 includes:

- an anti-aliasing/deflicker section 622,

- an RGB to YUV section, and

15

- a Y scale section which can be used during the process of copying the data from the embedded frame buffer (EFB) 702 to the external frame buffer (XFB) 113.

20 While the invention is directed to anti-aliasing, the second phase of which is performed by the anti-aliasing/deflickering section 622, a brief explanation of the other two sections in the copy pipeline is provided below to give a more complete understanding of the exemplary embodiment.

A luma/chroma (YUV) format stores the same visual quality pixel as RGB,

25 but requires only two-thirds of the memory. Therefore, during the copy operation, the RGB format in the EFB is converted to a YUV format in the XFB, in order to

reduce the amount of main memory used for the external frame buffer (XFB). This conversion is done by the RGB to YUV section 624.

The Y scale section 626 in the copy pipeline 620 enables arbitrary scaling of a rendered image in the vertical direction. Horizontal scaling is done during video display. A Y scale factor is defined in the API and determines the number of lines that will be copied, and can be used to compute the proper XFB size.

While not shown in Figure 13, a gamma correction section my also be provided in the copy pipeline 620 between, for example, the anti-aliasing/deflicker section and the RGB to YUV section. Gamma correction is used to correct for the non-linear response of the eye (and sometimes the monitor) to linear changes in color intensity values. Three choices of gamma may be provided (such as 1.0, 1.7 and 2.2). The default gamma is preferably 1.0 and is set in, for example, a GXInit command in the API.

The anti-aliasing/deflickering section 622 of the copy pipeline 620 applies a 7 tap vertical filter 628 having programmable weightings (W0-W6). A sub-pixel weighted area sampling is used for anti-aliasing. The support for the vertical filter is a three-vertical-pixel area. An exemplary support area for this filter is shown in Figure 14. When determining color for a current pixel N in anti-aliasing mode, super-samples in the pixel immediately above the current pixel (N-1), and super-samples in the pixel immediately below the current pixel (N+1), as well as super-samples in the current pixel (N) are used. Thus, a fragment's influence is not restricted to a single pixel, but rather is applied to other pixels too using a weighted 3-pixel vertical filter.

Example Copy Out Command

The EFB source and destination of the copy operation is specified using an exemplary API function as follows:

<u>GXCopyFBToDisplay</u>

<u>Argument</u>

  u16 SrcLeft   //Upper-Left coordinate of the source rectangle

  u16 SrcTop

  u16 SrcWidth //Width, in pixels, of the source rectangle

  u16 SrcHeight //Height, in pixels, of the source rectangle

  Void*DstBase //Address of destination buffer in memory

  u16 DstStride //Stride, in multiple of 32B, of destination buffer

  GXBool Clear //enable clearing color and Z frame buffers

This function copies the contents of the embedded frame buffer (EFB) to the display buffer 113 in main memory. By the term "copy out" we don't mean simply a transfer of all the information; rather, we mean that the contents of the embedded frame buffer are read out, further processed (e.g., filtered, resampled, scaled, etc.) and that the resulting data is then sent elsewhere (e.g., to an external point sample type frame buffer). The origin of the rectangle is at X=SrcLeft and Y=SrcTop. The Clear flag enables clearing of the color and z buffer to the current clear color and z values. The clearing of the embedded frame buffer preferably occurs simultaneously with the copy operation.

As shown in Figure 14, when in anti-aliasing mode in the example embodiment, the blending filter 628 uses all of the super-samples from the current pixel, some super-samples from the pixel immediately above the current pixel and some samples from the pixel immediately below the current pixel. Preferably, the farthest sample from the current pixel within each of the two surrounding pixels (i.e. N-1 and N+1) is not used in the filtering operation. While the three pixel support for the filter has nine samples, only seven of the nine samples are used in the blending operation in the example embodiment, as shown in Figure 14.

The example anti-alias filter provides a one-dimensional (i.e., 1x2 or vertical) filtering "aperture" encompassing all of the subpixels in the pixel whose color is being developed plus some additional subpixels of the pixels immediately above and below the current pixel. See Figure 15. Since the example

5 implementation filters vertically anyway for flicker reduction, this is a very inexpensive way of doubling the antialiasing quality for vertical edges. Even for non-interlaced displays, most viewers are probably willing to sacrifice a certain amount of filtering of the image in exchange for better anti-aliasing.

The resulting vertical filter output provides a single screen pixel color value

10 (RGB) for copying into the external frame buffer and display on display device 56. This vertical filtering operation thus acts as a low-pass filter stage as well as a resampler that resamples at the resolution of display 56. In this particular example, the neighboring pixels to the left and right of the current pixel Pixc do not contribute to the screen pixel output. In other implementations, however, a

15 horizontal filter or a combination vertical and horizontal filter could be used. In the example embodiment, horizontal resampling may be activated for other purposes to provide resampling in both dimensions.

Figure 16 shows an example vertical filter structure that implements the functions shown in Figure 14.

20 In the example embodiment, a respective weighting coefficient (W0-W6) is applied to each of the seven samples being vertically filtered, and then the weighted samples are added (blended) together to obtain the final pixel color (N') for the current screen pixel. The respective weighting coefficients (W0-W6) are programmable in this example. The following is an example of an API function

25 that can be used to set these programmable weighting coefficients.

GXSetAAFilter

Argument:

u8   Coefficients[7]   //filter coefficients in multiples of 1/64

This function sets the vertical filtering coefficients for anti-aliasing during copy-out from the embedded frame buffer. The filter coefficients are 6-bit

5   numbers given in multiples of 1/64. The filter coefficients are applied to vertical lines as the image is copied out to the external frame buffer. The same coefficients can be used for de-flickering the image when it is copied out (i.e. when anti-aliasing is not enabled).

Figure 11 shows exemplary registers 630a and 630b for storing the seven

10   weighting coefficients for use in the filtering operation of Figure 14. While the weighting coefficients are programmable, example weightings for anti-aliasing with this vertical filter useful with the multisample spatial distribution pattern shown in Figure 10 are as follows:

W0 =1/12, W1=1/6, W2=1/6, W3=1/6, W4=1/6, W5=1/6 and W6=1/12

15   As discussed above, different filter weights and/or configurations can be used with different numbers of multisamples per pixel, different multisample spatial distributions, and different reconstruction filter apertures.

Anti-Alias Buffering

Figure 16 shows an example anti-aliasing copy out buffering operation used

20   to copy out the contents of embedded frame buffer 702 into an external frame buffer within main memory 115. In this example, the embedded frame buffer 702 is organized into tiles that are 32 pixels wide by 32 pixels high. Copying out is done a tile-by-tile basis. However, as shown in the shaded region of Figure 17, the copy out operation needs one pixel from the immediately uppermost adjacent

25   neighboring tile and one pixel from the immediately lowermost adjacent neighboring tile in order to perform the vertical anti-aliasing filter function

described above. Therefore, in the preferred embodiment, the copy out operation is performed by reading out a 34-pixel high by 32-pixel wide tile. In the example embodiment (which also includes a vertical scaling capability), copying is performed first in the y direction and then in the x direction to facilitate vertical

5   zooming.

In order to avoid the use of full line buffers, the copy operation uses anti-aliasing (AA) buffering, wherein the copy is performed in strips of 32 pixels wide (X axis). The data-path for the strip-buffers in this exemplary AA buffering is shown in the block diagram of Figure 18. For each strip, two extra pixels are read

10   in the left and right, for a total buffer size of 36. Data from two scan-lines are stored in four buffers in the following order:

- buffer0 holds horizontal pixel pairs with x[1]=0, y[0]=0 (bankA)
- buffer1 holds horizontal pixel pairs with x[1]=0, y[0]=1 (bankB)
- buffer0 holds horizontal pixel pairs with x[1]=1, y[0]=0 (bankA)

15   - buffer1 holds horizontal pixel pairs with x[1]=1, y[0]=1 (bankB)
- The third scan-line comes from live data from the embedded frame buffer. The shifters 632 provide RGB pixel data from the separate lines to the respective AA filter 628.

Another aspect to anti-aliasing in the preferred embodiment is that the

20   maximum screen resolution that can be supported in one pass drops down to 640 pixels by 288 pixels. If higher display resolution is required, then due to the size limitation of the embedded frame buffer 702, multiple passes through the scene may need to be performed. At the end of each pass, the image is copied out into main memory. For example, a scene can be rendered in two passes by rendering

25   the top half of the scene into the embedded frame buffer; copying out the top half of the scene into the top half of an external frame buffer; rendering the bottom half

of the scene into the embedded frame buffer; and copying out the bottom half of the scene into the external frame buffer before finally displaying the entire scene from the external frame buffer. Such a multiple-pass operation is slower than a single-pass operation but reduces the amount of embedded frame buffer required

5   on chip. As memory becomes cheaper in the future, it will be possible to incorporate additional embedded frame buffer memory (e.g., four megabytes, eight megabytes or more) on chip such that even high resolution images may be rendered and copied out in a single pass.

Example De-Flickering Filtering During Copy Out

10   The same vertical filter can be used during copy-out in a non-anti-aliasing mode to achieve a de-flickering function using point sampled pixels. De-flickering is typically used to solve two problems: 1) to eliminate flickering of thin one-pixel tall horizontal lines for interlaced video display (a one pixel tall horizontal line will flicker at 30Hz as the TV interlaced video shows this line every other field); and 2)

15   to provide simple anti-aliasing by rendering, for example, at 60Hz 480 lines in the frame buffer and deflickering to 240 lines for interlaced display. This is basically 2-sample super-sampling.

In this example of the non-anti-aliasing mode (de-flickering mode), the sample patterns are not programmable. Thus, in this example, the hardware uses

20   only the center of the pixel as the sample locations. Thus, the programmable super-sample locations are ignored in this exemplary mode. An example blending filter 628a for de-flickering is shown in Figure 19. The weighting coefficients (coeff0-coeff6) are programmable and can correspond to the weightings (W0-W6) in the anti-aliasing filter shown in Fig. 14. As shown in Figure 19, the vertical

25   filter 628a in de-flickering mode uses three inputs (center only) from the current pixel and two inputs (center only) from each of the two vertically neighboring

pixels (N-1 and N+1), thereby obtaining the seven values for the filtering operation. The programmable weighting coefficients (coeff0-coeff6) are applied to the seven samples, and then the results are added to obtain the final pixel color (N'). For de-flickering, coeff0 and coeff1 are preferably set to the same value,

5   coeff2, coeff3 and coeff4 are preferably set to the same value, and coeff5 and coeff6 are preferably set to the same value. These values may be defined in the same manner as W0-W6 described above with respect to the anti-aliasing filter 628. It is noted that if it is desired to have N' be the same as N, coeff0, coeff1, coeff5 and coeff6 can be set to zero, and the remaining weights (coeff2, coeff3 and

10   coeff4) can be set so that they total one (1). This will enable the filter to output a value of N' that is the same as the value of the single sample in pixel N.

It is noted that the location of the point sample in each pixel may be programmable in another embodiment of the invention. In other words, the invention is not limited to using the center of the pixel as the point sampled

15   location. This location may be set by hardware or programmable so as to vary from one pixel to the next in order to constitute a specific sampling pattern between adjacent pixels, as described above with respect to the supersampling or anti-aliasing mode. For example, a specific pattern for the point sample locations may be set on a quad-by-quad basis or otherwise in order to improve the anti-

20   aliasing achieved in this point sampled embodiment. As explained in detail above for the anti-aliasing embodiment, the extent of the reconstruction filter (deflicker filter) can then be increased in the vertical, horizontal or both direction(s), to greater than 1x1 for the purpose of improving anti-aliasing. Thus, by using a particular point sample location pattern (i.e. jittering between pixels) and a

25   particular filter configuration that share some point samples from neighboring pixels, better anti-aliasing can be achieved in accordance with the instant invention.

De-flickering can be optionally performed as described above to convert a frame to a field. Preferably, the de-flickering filter and AA filter are shared. The four strip buffers used in the AA data path (see Figure 17) are also used to store quad strips. An exemplary block diagram of the data-path for de-flicker buffering

5   is shown in Figure 20. The shifter section 632a provides the three RGB pixel data from the three separate lines to the AA-filter. This is a three-tap filter with programmable coefficients.

Other Example Compatible Implementations

Certain of the above-described system components 50 could be implemented

10   as other than the home video game console configuration described above. For example, one could run graphics application or other software written for system 50 on a platform with a different configuration that emulates system 50 or is otherwise compatible with it. If the other platform can successfully emulate, simulate and/or provide some or all of the hardware and software resources of

15   system 50, then the other platform will be able to successfully execute the software.

As one example, an emulator may provide a hardware and/or software configuration (platform) that is different from the hardware and/or software configuration (platform) of system 50. The emulator system might include

20   software and/or hardware components that emulate or simulate some or all of hardware and/or software components of the system for which the application software was written. For example, the emulator system could comprise a general purpose digital computer such as a personal computer, which executes a software emulator program that simulates the hardware and/or firmware of system 50.

25   Some general purpose digital computers (e.g., IBM or MacIntosh personal computers and compatibles) are now equipped with 3D graphics cards that provide

3D graphics pipelines compliant with DirectX or other standard 3D graphics command APIs. They may also be equipped with stereophonic sound cards that provide high quality stereophonic sound based on a standard set of sound commands. Such multimedia-hardware-equipped personal computers running

5 emulator software may have sufficient performance to approximate the graphics and sound performance of system 50. Emulator software controls the hardware resources on the personal computer platform to simulate the processing, 3D graphics, sound, peripheral and other capabilities of the home video game console platform for which the game programmer wrote the game software.

10 Figure 21A illustrates an example overall emulation process using a host platform 1201, an emulator component 1303, and a game software executable binary image provided on a storage medium 62. Host 1201 may be a general or special purpose digital computing device such as, for example, a personal computer, a video game console, or any other platform with sufficient computing

15 power. Emulator 1303 may be software and/or hardware that runs on host platform 1201, and provides a real-time conversion of commands, data and other information from storage medium 62 into a form that can be processed by host 1201. For example, emulator 1303 fetches "source" binary-image program instructions intended for execution by system 50 from storage medium 62 and

20 converts these program instructions to a target format that can be executed or otherwise processed by host 1201.

As one example, in the case where the software is written for execution on a platform using an IBM PowerPC or other specific processor and the host 1201 is a personal computer using a different (e.g., Intel) processor, emulator 1303 fetches

25 one or a sequence of binary-image program instructions from storage medium 62 and converts these program instructions to one or more equivalent Intel binary-image program instructions. The emulator 1303 also fetches and/or generates

graphics commands and audio commands intended for processing by the graphics and audio processor 114, and converts these commands into a format or formats that can be processed by hardware and/or software graphics and audio processing resources available on host 1201. As one example, emulator 1303 may convert

5   these commands into commands that can be processed by specific graphics and/or or sound hardware of the host 1201 (e.g., using standard DirectX, OpenGL and/or sound APIs).

One example way to implement anti-aliasing on an emulator is not to implement it at all; in other words, an emulator might entirely ignore or "stub"

10   calls directed to turning anti-aliasing on and off. Another possibility is to activate the same or different form of anti-aliasing in response to activation of the anti-aliasing API calls discussed above. An emulator running on a standard personal computer with a standard graphics card may not have an embedded frame buffer or the hardware-based anti-aliasing filtering arrangements and the particular frame

15   buffer formats discussed above. Accordingly, anti-aliasing could be performed in an entirely different way – or it could be performed in essentially the same way under software control where the software operates on the contents of the standard frame buffer to provide the filtering discussed above. Certain advanced graphics cards supposedly support anti-aliasing in hardware, but at the present time, the

20   more usual approach is to perform anti-aliasing under software control. However, as standard graphics cards become more advanced and anti-aliasing support in hardware becomes more available, an emulator might use a different anti-aliasing approach to emulate the anti-aliasing called for by an application written to run on the system shown in Figure 1.

25   An emulator 1303 used to provide some or all of the features of the video game system described above may also be provided with a graphic user interface (GUI) that simplifies or automates the selection of various options and screen

modes for games run using the emulator. In one example, such an emulator 1303 may further include enhanced functionality as compared with the host platform for which the software was originally intended.

Figure 21 illustrates an emulation host system 1201 suitable for use with

5   emulator 1303. System 1201 includes a processing unit 1203 and a system memory 1205. A system bus 1207 couples various system components including system memory 1205 to processing unit 1203. System bus 1207 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System

10   memory 1207 includes read only memory (ROM) 1252 and random access memory (RAM) 1254. A basic input/output system (BIOS) 1256, containing the basic routines that help to transfer information between elements within personal computer system 1201, such as during start-up, is stored in the ROM 1252. System 1201 further includes various drives and associated computer-readable

15   media. A hard disk drive 1209 reads from and writes to a (typically fixed) magnetic hard disk 1211. An additional (possible optional) magnetic disk drive 1213 reads from and writes to a removable "floppy" or other magnetic disk 1215. An optical disk drive 1217 reads from and, in some configurations, writes to a removable optical disk 1219 such as a CD ROM or other optical media. Hard disk

20   drive 1209 and optical disk drive 1217 are connected to system bus 1207 by a hard disk drive interface 1221 and an optical drive interface 1225, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules, game programs and other data for personal computer system 1201. In other configurations, other

25   types of computer-readable media that can store data that is accessible by a computer (e.g., magnetic cassettes, flash memory cards, digital video disks,

Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like) may also be used.

A number of program modules including emulator 1303 may be stored on the hard disk 1211, removable magnetic disk 1215, optical disk 1219 and/or the ROM 1252 and/or the RAM 1254 of system memory 1205. Such program modules may include an operating system providing graphics and sound APIs, one or more application programs, other program modules, program data and game data. A user may enter commands and information into personal computer system 1201 through input devices such as a keyboard 1227, pointing device 1229, microphones, joysticks, game controllers, satellite dishes, scanners, or the like. These and other input devices can be connected to processing unit 1203 through a serial port interface 1231 that is coupled to system bus 1207, but may be connected by other interfaces, such as a parallel port, game port Fire wire bus or a universal serial bus (USB). A monitor 1233 or other type of display device is also connected to system bus 1207 via an interface, such as a video adapter 1235.

System 1201 may also include a modem 1154 or other network interface means for establishing communications over a network 1152 such as the Internet. Modem 1154, which may be internal or external, is connected to system bus 123 via serial port interface 1231. A network interface 1156 may also be provided for allowing system 1201 to communicate with a remote computing device 1150 (e.g., another system 1201) via a local area network 1158 (or such communication may be via wide area network 1152 or other communications path such as dial-up or other communications means). System 1201 will typically include other peripheral output devices, such as printers and other standard peripheral devices.

In one example, video adapter 1235 may include a 3D graphics pipeline chip set providing fast 3D graphics rendering in response to 3D graphics commands issued based on a standard 3D graphics application programmer interface such as

Microsoft's DirectX 7.0 or other version. A set of stereo loudspeakers 1237 is also connected to system bus 1207 via a sound generating interface such as a conventional "sound card" providing hardware and embedded software support for generating high quality stereophonic sound based on sound commands provided by

5   bus 1207. These hardware capabilities allow system 1201 to provide sufficient graphics and sound speed performance to play software stored in storage medium 62.

     While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be

10   understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the scope of the appended claims.

We Claim:

1   1. In a graphics system including graphics circuits coupled to an embedded

2   frame buffer, an anti-aliasing method comprising:

3   (a) rendering a multisampled data representation in the embedded

4   frame buffer;

5   (b) storing the rendered multisampled data representation in the

6   embedded frame buffer; and

7   (c) resampling the embedded frame buffer contents to provide an anti-

8   aliased image.

1   2. The method of claim 1, further including defining a sample pattern for

2   use in rendering the multisampled data representation, and using a reconstruction

3   filter during resampling of the embedded frame buffer, wherein the reconstruction

4   filter uses multisamples from more than one pixel region to obtain data for a

5   resulting pixel.

1   3. The method of claim 2, wherein a particular support area for the

2   reconstruction filter is determined based on the sample pattern.

1   4. The method of claim 1, further including varying a sample pattern for

2   multisamples among adjacent pixels, and using a reconstruction filter during

3   resampling having a support region that extends beyond a single pixel

1   5. The method of claim 4, further including defining a particular support

2   region for the reconstruction filter based on a particular sample pattern for the

3   multisamples.

1   6. In a graphics system of the type that generates an image comprising

2 plural pixels, an anti-aliasing method comprising:

3   (a) generating a multisampled data representation of an image having

4 plural samples associated with each of the plural pixels; and

5   (b) resampling the multisampled data representation, wherein the

6 resampling includes blending at least one of the plural samples into plural image

7 pixels.

1   7. The method of claim 6, further including storing the multisampled data

2 representation in an embedded frame buffer, and further wherein the resampling

3 includes resampling from the embedded frame buffer.

1   8. The method of claim 6, further including a sampling pattern having a

2 non-uniform spatial distribution for the plural samples within neighboring pixels.

1   9. The method of claim 7, further including using a blending filter for the

2 blending which has a support region that is greater than a single pixel.

1   10. The method of claim 8, further including using a blending filter for the

2 blending which has a support region that is greater than a single pixel and is

3 defined based on the sampling pattern.

1   11. The method of claim 10, wherein the support region covers a current

2 pixel and at least a portion of at least two neighboring pixels to the current pixel.

1   12. An anti-aliasing method, comprising:

2   (a) providing plural supersamples within each pixel of a pixel array;

3   (b) varying the spatial distribution of the supersamples within

4 neighboring pixels of the pixel array;

5   (c) applying, to the array, an anti-aliasing filter having a pixel aperture

6 including a current pixel and at least one of the supersamples from at least two

7 neighboring pixels to the current pixel; and

8   further including storing the pixel array in an embedded frame buffer, and

9 applying the anti-aliasing filter during a copy out operation from the embedded

10 frame buffer to an external destination.

1   13. The method of claim 12, wherein the varying of the supersamples

2 defines a sample pattern, and further including defining the aperture of the

3 antialiasing filter based on the sample pattern

1   14. The method of claim 13, wherein the sample pattern repeats on a pixel

2 quad basis.

1   15. The method of claim 14, wherein the sample pattern is different for each

2 pixel in a pixel quad.

1   16. In a graphics chip including an embedded frame buffer, an anti-aliasing

2 method comprising:

3   (a) storing a supersampled image in the embedded frame buffer;

4   (b) transferring the stored image from the embedded frame buffer to

5 an off-chip destination; and

6   (c) in the process of transferring the image, resampling the image to

7 provide an anti-aliased version of the image.

1   17. The method of claim 16, further including defining a sampling pattern

2 for use in generating the supersampled image, wherein the sampling pattern varies

3 between adjacent pixels of the image.

1 18. The method of claim 17, wherein the resampling includes using a

2 blending filter having a pixel aperture which is greater than one pixel.

1 19. The method of claim 18, further including defining the pixel aperture

2 based on the sampling pattern.

1 20. In a graphics system, a method of anti-aliasing super-sampled pixels,

2 comprising the steps of:

3 (a) defining, within an embedded frame buffer, super-sample locations for

4 each of a plurality of neighboring pixels of an image;

5 (b) assigning color data to each of said super-sample locations; and

6 (c) blending color data from at least two samples obtained from locations

7 defined in step (a) to provide a pixel final color value.

1 21. The method of claim 20, wherein the defining step (a) comprises

2 programming variable sample locations.

1 22. The method as in claim 20, wherein the defining step (a) comprises

2 defining three sample locations for each pixel in a 2 X 2 pixel quad.

1 23. The method as in claim 20, further including wherein the defining step

2 (a) comprises programming sample locations as x and y distances in units of one-

3 twelfth of a pixel.

1 24. The method as in claim 20, further including using a coverage mask to

2 enable/disable samples corresponding to locations defined in step (a), the coverage

3 mask being based at least in part on corresponding portions of each pixel that are

4 occupied by a primitive fragment; and wherein the coverage mask is further based

5 on depth comparisons of primitive fragments at the sample locations.

1   25.   The method as in claim 24, wherein the coverage mask comprises a

2   masking bit corresponding to each sample location in a quad of pixels.

1   26.   The method as in claim 20, wherein color data associated with pixels

2   is stored within a random access memory embedded within a graphics chip, and

3   step (c) is performed during an operation of transferring data from the embedded

4   random access memory to a memory external of the graphics chip.

1   27.   The method of claim 20, wherein the blending step (c) includes

2   assigning blending weights to one or more samples, and blending enabled samples

3   based at least in part on assigned weights.

1   28.   The method of claim 27, wherein the weights are assigned via an API

2   program function.

1   29.   The method of claim 28, wherein the weights are defined in multiples

2   of 1/64.

1   30.   The method of claim 20, wherein the blending step (c) includes

2   assigning weights for seven of the samples, and blending the seven samples based

3   at least in part on assigned weights.

1   31.   The method of claim 30, wherein the weights are assigned via an API

2   program function.

1   32.   The method of claim 20, wherein the blending step (c) includes

2   blending seven samples including three samples from a current pixel with two

3   samples taken from a pixel immediately above the current pixel and two samples

4   taken from a pixel immediately below the current pixel.

1   33.   The method of claim 20, wherein each super-sampled pixel is

2   represented in memory by at least three samples of 16-bit color data and three

3   samples of corresponding16-bit Z position data.

1   34.   A graphics system, an apparatus for anti-aliasing super-sampled

2   pixels, comprising:

3   means for programmably defining three sample locations for obtaining

4   super-sampled color data associated with a pixel for each of a plurality of

5   neighboring pixels;

6   coverage mask means to enable/disable samples corresponding to said

7   sample locations, the coverage mask means being based at least in part on

8   corresponding portions of each pixel that are occupied by rendered primitive

9   fragments; and

10   color data blending filter means for combining color data from at least two

11   super-sampled color data to provide a pixel final color value.

1   35.   The system of claim 34, wherein said blending filter means comprises

2   a means for computing a weighted average of samples.

1   36.   The system of claim 34, wherein said blending filter means comprises

2   a means for computing a weighted average of color data of at least three samples

3   corresponding to a current pixel and at least two samples corresponding to a pixel

4   immediately above the current pixel and at least two samples corresponding to a

5   pixel immediately below the current pixel.

1   37.   The system of claim 34, wherein the blending filter means further

2   comprises a weighting coefficient means for selectively weighting each sample of

3   color data for computing a weighted average of color data, the graphics system

4   including a means for programmably defining a weight coefficient associated with

5   each sample.

1   38.   In a graphics system, a method of providing full-scene anti-aliasing,

2   comprising the steps of:

3   (a) defining three super-sampled color data locations associated with a pixel

4   for each of a plurality of neighboring pixels;

5   (b) blending the three super-sampled color data locations within two super-

6   sampled color locations of a pixel immediately above the current pixel and two

7   super-sampled color locations of a pixel immediately below the current pixel; and

8   (c) displaying a pixel having a color corresponding to the blend.

1   39.   The method of claim 38, wherein the blending step (b) includes

2   assigning weights for the seven super-sampled color data locations, and computing

3   a weighted average based at least in part on assigned weights.

1   40.   In a graphics system, a method of anti-aliasing pixels wherein each

2   pixel is subdivided into a plurality of super-sampled portions identified by

3   locations programmably defined therein, comprising the steps of:

4   (a) defining a plurality of super-sampled locations for each of a plurality of

5   neighboring pixels;

6   (b) using coverage masks to develop color data for super-samples

7   corresponding to locations defined in step (a), the coverage masks being based at

8   least in part on corresponding portions of each pixel that are occupied by primitive

9   fragments; and

10   (c) blending color data from at least two selected super-samples obtained

11 from locations defined in step (a) during a copy-out operation to provide a filtered

12 pixel color value.

1   41. In a graphics system, a pixel data processing arrangement having a

2   multi-tap selectable-weight blending filter characterized by a vertically-arranged

3   multiple-pixel filter support region wherein one or more color data samples from a

4   plurality of vertically disposed pixels are blended to form a pixel color.

1   42.   In a graphics system, a pixel data processing arrangement for

2   providing full-scene anti-aliasing and/or de-flickering interlaced displays,

3   comprising:

4   a frame buffer containing super-sampled pixel data for a plurality of pixels;

5   a plurality of scan-line buffers connected to receive super-sampled pixel

6   color data from the frame buffer; and

7   a multi-tap selectable-weight blending filter coupled to the scan-line buffers,

8   the blending filter characterized by a vertically-arranged multiple-pixel filter

9   support region wherein one or more color data samples from a plurality of

10   vertically disposed pixels are blended to form a pixel color.

1   43. An apparatus for anti-aliasing as set forth in claim 42, wherein pixel

2   data in the frame buffer also includes depth (Z data) information.

1   44.   An arrangement that anti-aliases super-sampled pixels comprising:

2   an embedded frame buffer storing three super-sample locations within each

3   pixel of a pixel array, each said super-sample location having a corresponding

4   color value; and

5   a one-dimensional color data blending filter that blends the three super-

6   sample color values with super-sample color values of adjacent neighboring pixels

7   while information within the embedded frame buffer is being transferred to a

8   destination.

1   45. The arrangement of claim 44, wherein the embedded frame buffer stores

2   no more than three super-sample locations within each pixel.

1   46. The arrangement of claim 44, wherein the filter blends super-sample

2   color values corresponding to each pixel with super-sample color values

3   corresponding to at least one further neighboring pixel.

1   47. The arrangement of claim 44, wherein the filter blends super-sample

2   color values corresponding to three vertically aligned pixels to produce a screen

3   pixel output.

1   48.   An anti-aliasing method comprising:

2   programmably defining plural super-sampled locations within at least one

3   screen pixel, each said super-sampled location having a corresponding color value;

4   and

5   blending said super-sampled color values using a vertical filter during a

6   copy-out operation from an embedded frame buffer to an external frame buffer.

1   49. Within a pixel quad having first, second, third and fourth pixels and a

2   quad center, a method of defining an optimal set of three super-sampling locations

3   for anti-aliasing, said method comprising:

4   (a) defining a first set of super-sample locations for a first pixel in the pixel

5   quad at the following coordinates (range 1-12) relative to the quad center:

6   (12,11)

7   (4,7)

8   (8,3);

9   (b) defining a second set of super-sample locations for a second pixel in the

10   pixel quad at the following coordinates (range 1-12) relative to the quad center:

11   (3,11)

12   (11,7)

13   (7,3);

14   (c) defining a third set of super-sample locations for a third pixel in the pixel

15   quad at the following coordinates (range 1-12) relative to the quad center:

16   (2,2)

17   (10,6)

18   (6,10);

19   (d) defining a fourth set of super-sample locations for a fourth pixel in the

20   pixel quad at the following coordinates (range 1-12) relative to the quad center:

21   (9,2)

22   (1,6)

23   (5,6);

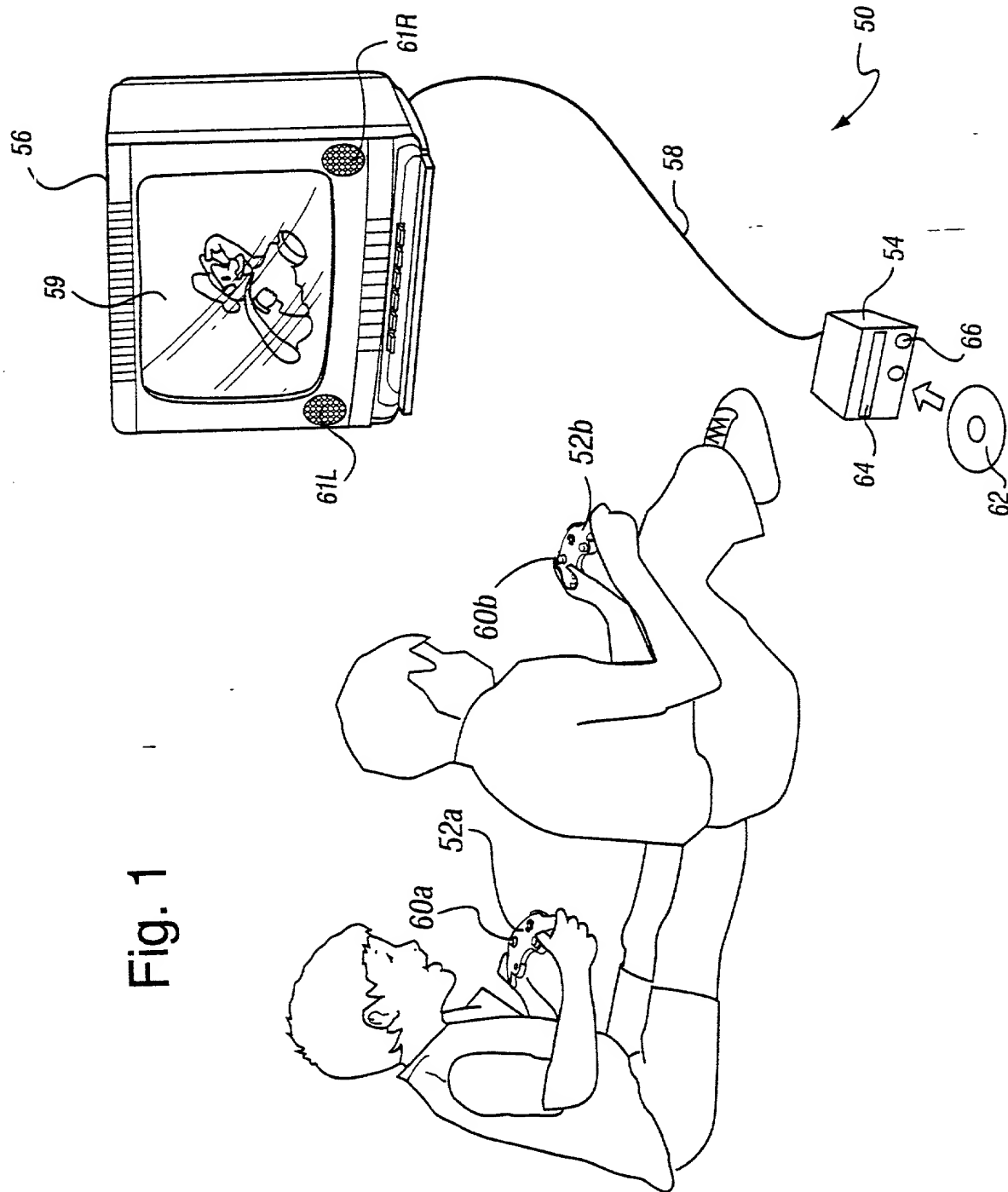24   (e) using a resampling filter having a support area that uses three

25   supersamples from a current pixel, two super-samples from a pixel immediately

26   above the current pixel, and two samples from a pixel immediately below the
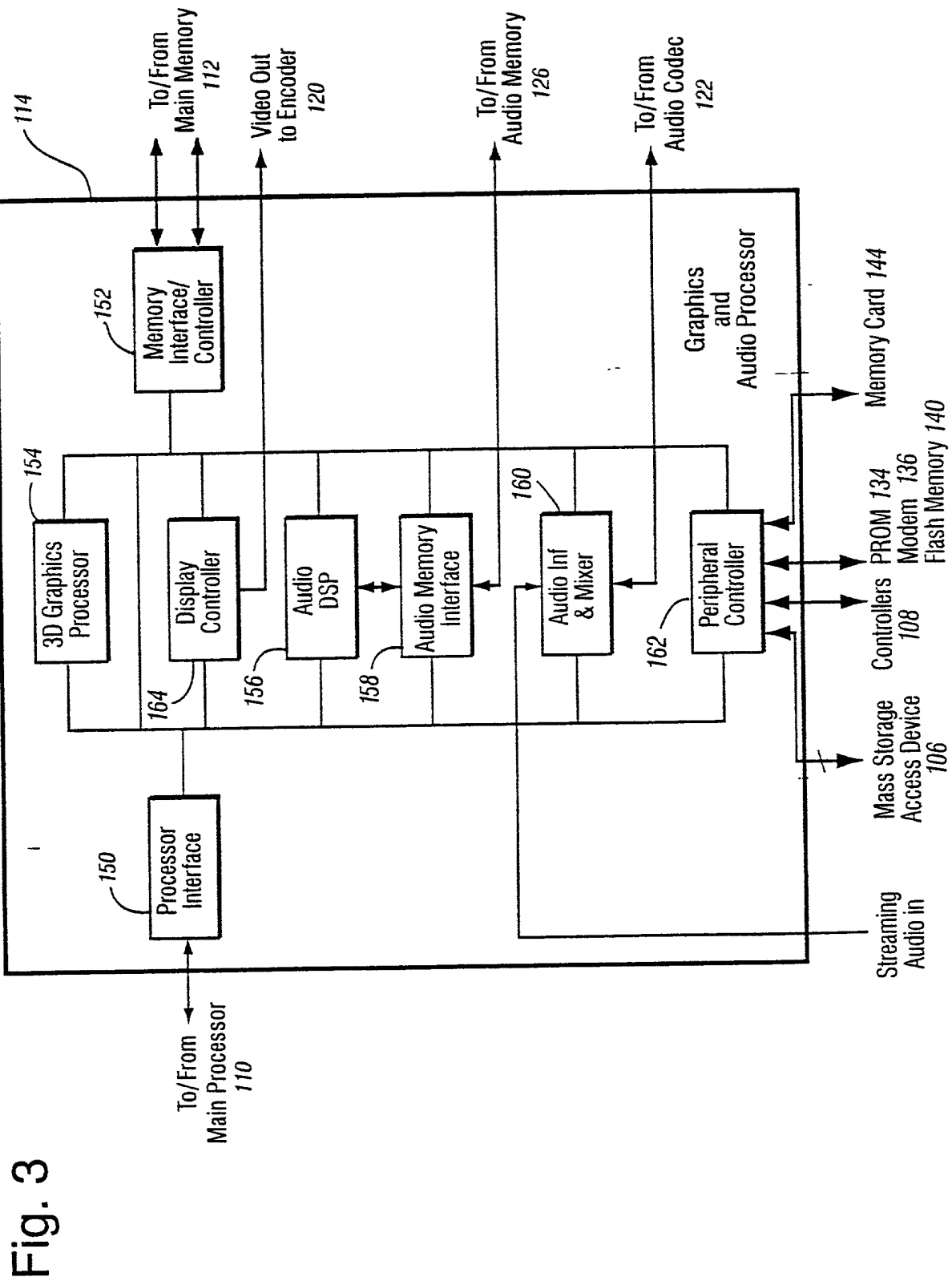
27   current pixel; and

28   (e) using respective weighting coefficients in the resampling filter

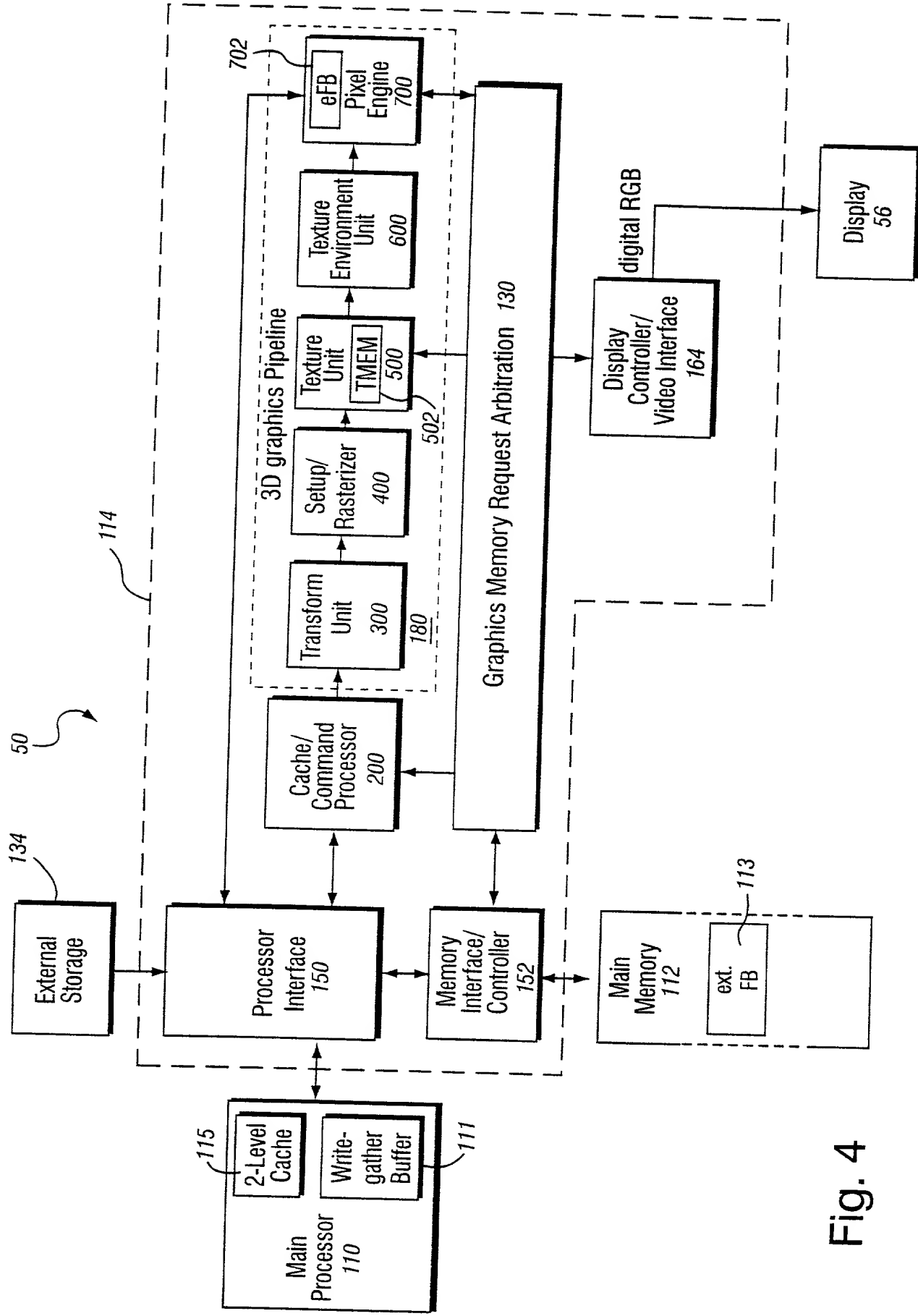29   having the following values: 1/12, 1/6, 1/6, 1/6, 1/6, 1/6, 1/12.

Abstract Of The Disclosure

A graphics system including a custom graphics and audio processor produces exciting 2D and 3D graphics and surround sound. The system includes a graphics and audio processor including a 3D graphics pipeline and an audio digital signal processor. The system achieves highly efficient full-scene anti-aliasing by implementing a programmable-location super-sampling arrangement and using a selectable-weight vertical-pixel support area blending filter. For a 2X2 pixel group (quad), the locations of three samples within each super-sampled pixel are individually selectable. A twelve-bit multi-sample coverage mask is used to determine which of twelve samples within a pixel quad are enabled based on the portions of each pixel occupied by a primitive fragment and any pre-computed z-buffering. Each super-sampled pixel is filtered during a copy-out operation from a local memory to an external frame buffer using a pixel blending filter arrangement that combines seven samples from three vertically arranged pixels. Three samples are taken from the current pixel, two samples are taken from a pixel immediately above the current pixel and two samples are taken from a pixel immediately below the current pixel. A weighted average is then computed based on the enabled samples to determine the final color for the pixel. The weight coefficients used in the blending filter are also individually programmable. De-flickering of thin one-pixel tall horizontal lines for interlaced video displays is also accomplished by using the pixel blending filter to blend color samples from pixels in alternate scan lines.

Fig. 1

Fig. 2

Fig. 3

To/From
Main Processor
110

Processor
Interface
150

3D Graphics
Processor
154

Display
Controller
164

Audio
DSP
156

Audio Memory
Interface
158

Audio Inf
& Mixer
160

Peripheral
Controller
162

Memory
Interface/
Controller
152

To/From
Main Memory
112

Video Out
to Encoder
120

To/From
Audio Memory
126

To/From
Audio Codec
122

114

Graphics
and
Audio Processor

Streaming
Audio in

Mass Storage
Access Device
106

Controllers
108

PROM 134
Modem 136
Flash Memory 140

Memory Card 144

Fig. 4

Fig. 5 EXAMPLE GRAPHICS PROCESSOR FLOW

START

Using coverage masks, render and rasterize supersampled image
(three subpixels per pixel at programmable locations) into embedded frame buffer
550

During copy-out operation, filter supersampled image by blending the three subpixels
per pixel with certain subpixels of neighboring pixels using programmable weighting
552

Display copied out image on display device
554

Fig. 6
Example
anti-aliasing process

END

START

Programmable control
registers (samples specified
in uints of 1/12 pixel)
550a-1

Define three multi-sample
locations per pixel for
a current 2 x 2 quad
550a

Edge rasterizer/
z-computations
550b-1

Create a multi-sample coverage mask
for enabling/disabling samples based
on portion of pixel occupied by a
primitive and primitive depth
550b

Programmable
control registers
(weight coefficients)
552-a

Blend color data from enabled samples
using programmed weight coefficients
from three vertically aligned neighboring pixels
during copy-out from local memory
to external frame buffer
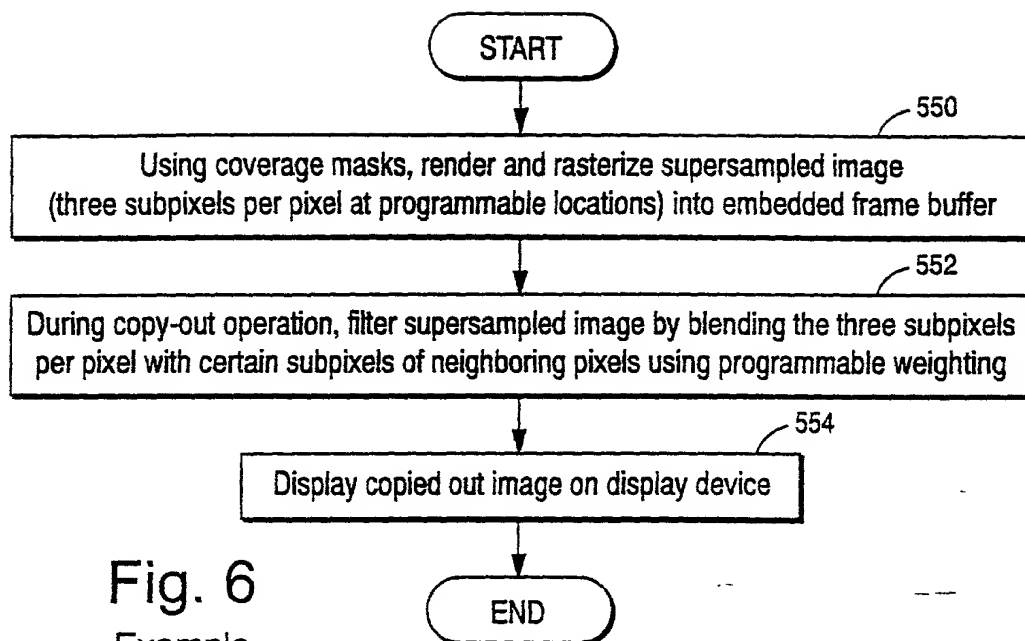552

Display
pixel
color
554

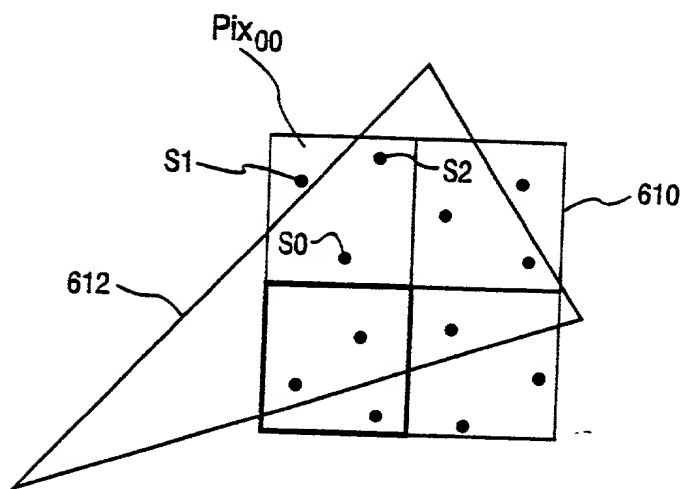## Fig. 6A
Example FSAA method flowchart

Fig. 7
(Primitive and super-
sampled piexel quad)

Fig. 8
(Sampling pattern)
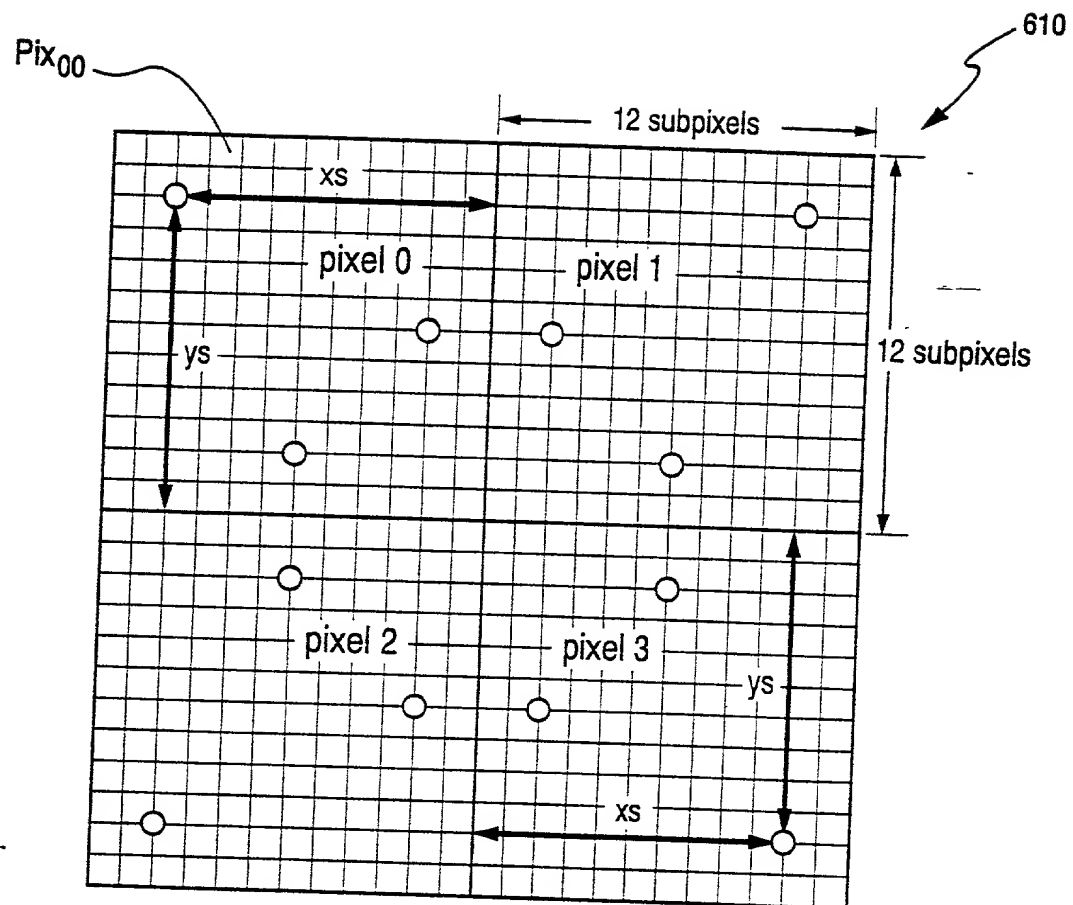
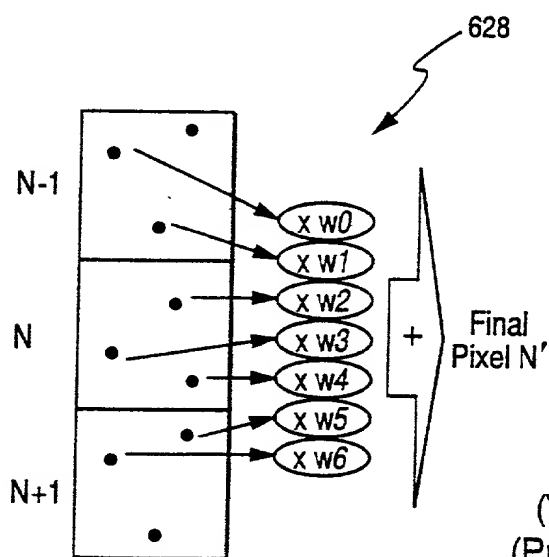**Fig. 9** (Super-sample locations in units of 1/12 pixel)

Fig. 10 (Preferred sampling pattern)

618

MODE (0x00)	ntex	ncol		ntev	

reject_en   ms_en   flat_en

LOC0 (0x01)	xs_{00}	ys_{00}	xs_{01}	ys_{01}	xs_{02}	ys_{02}	**616a**
LOC1 (0x02)	xs_{10}	ys_{10}	xs_{11}	ys_{11}	xs_{12}	ys_{12}	**616b**
LOC2 (0x03)	xs_{20}	ys_{20}	xs_{21}	ys_{21}	xs_{22}	ys_{22}	**616c**
LOC3 (0x04)	xs_{30}	ys_{30}	xs_{31}	ys_{31}	xs_{32}	ys_{32}	**616d**

vfilter r0	coeff0	coeff1	coeff2	coeff3	**630a**
vfilter r1	coeff4	coeff5	coeff6	unused	**630b**

Fig. 11 (Programmable control registers for setting sample locations & filter coefficients)

S2   S1   S0

| sample2 | sample1 | sample0 |

614

| pix$_{11}$ | pix$_{10}$ | pix$_{01}$ | pix$_{00}$ |

Fig. 12
(Super-sampling for current quad-coverage mask)

620

702

113

| EFB
(Enbedded
frame buffer) | | antialias
deflicker

filter state | RGB to YUV | Y scale

scale | XFB
(External
frame buffer) |

622   628   624   626

copy pipeline

Fig. 13 (Copy-out pipeline)

628

N-1

x w0
x w1
x w2
x w3
x w4
x w5
x w6

N

N+1

+   Final
Pixel N'

Fig. 14
(Vertical Filter Blending)
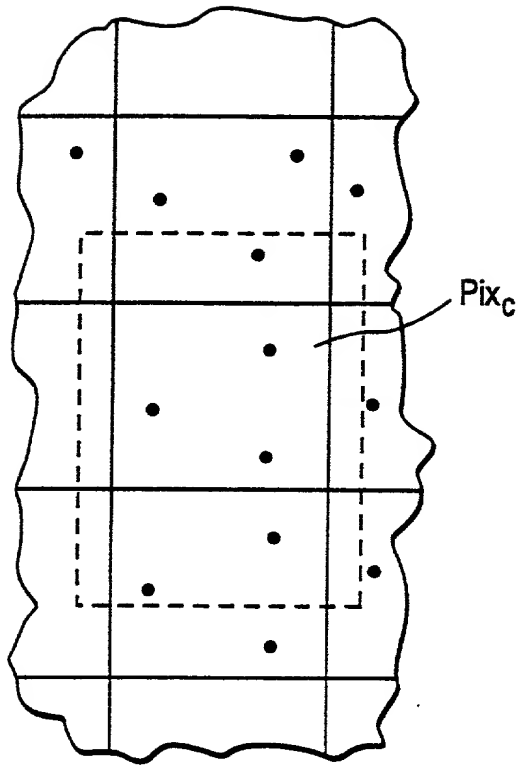(Programmable 7-tap filter)

Fig. 15

Example vertical filter aperture

Fig. 17

Example AA copy out buffering

Fig. 16
(Example Vertical Filter Structure)

Subpixel color values of neighboring pixel above { S1, S2 }

Subpixel color values of current pixel { S0, S1, S2 }

Subpixel color values of neighboring pixel below { S0, S1 }

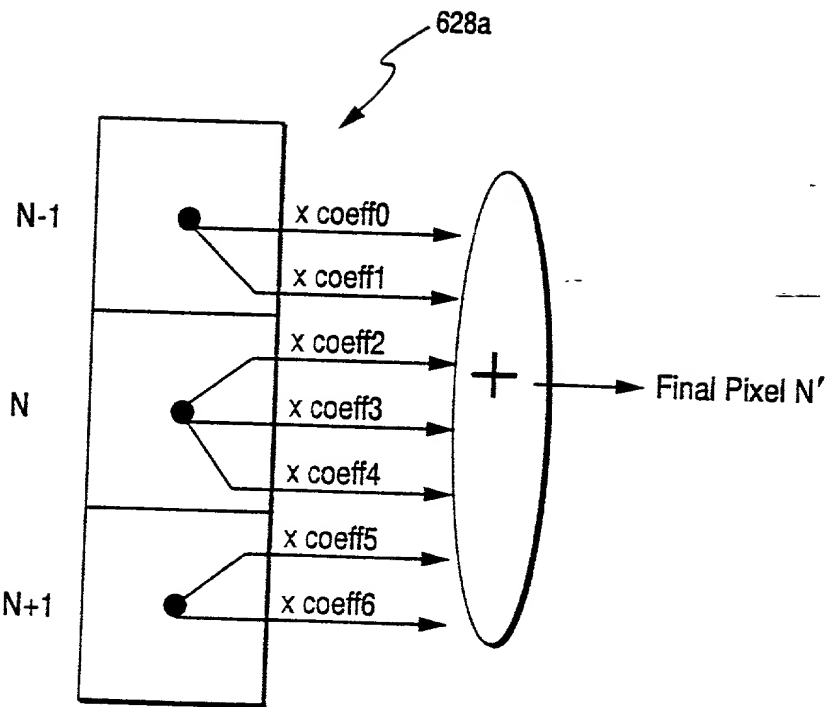WG0, WG1, WG2, WG3, WG4, WG5, WG6

Screen Pixel Color Value

Fig. 18
(AA buffering)
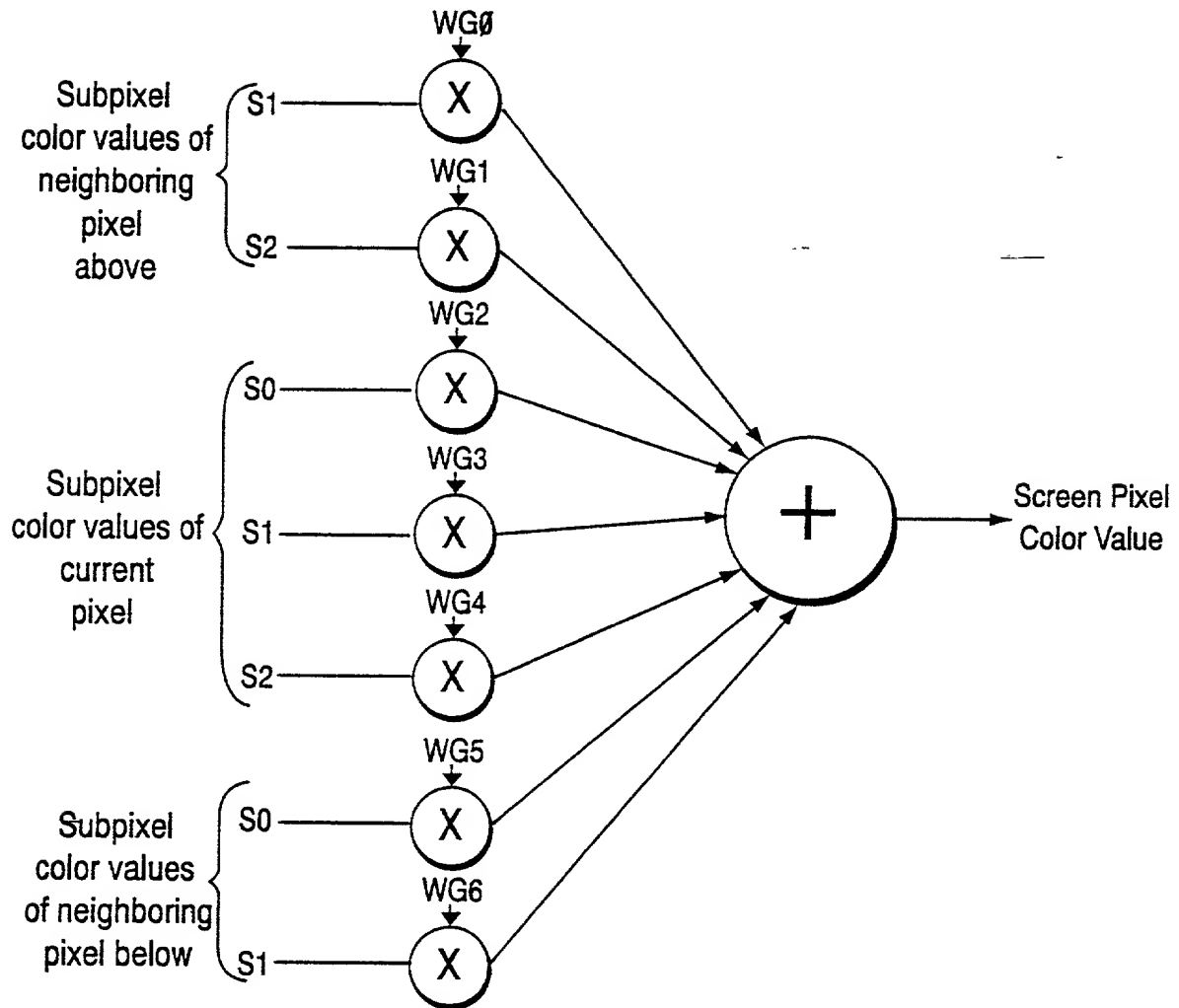
Fig. 19

Example de-flickering filter
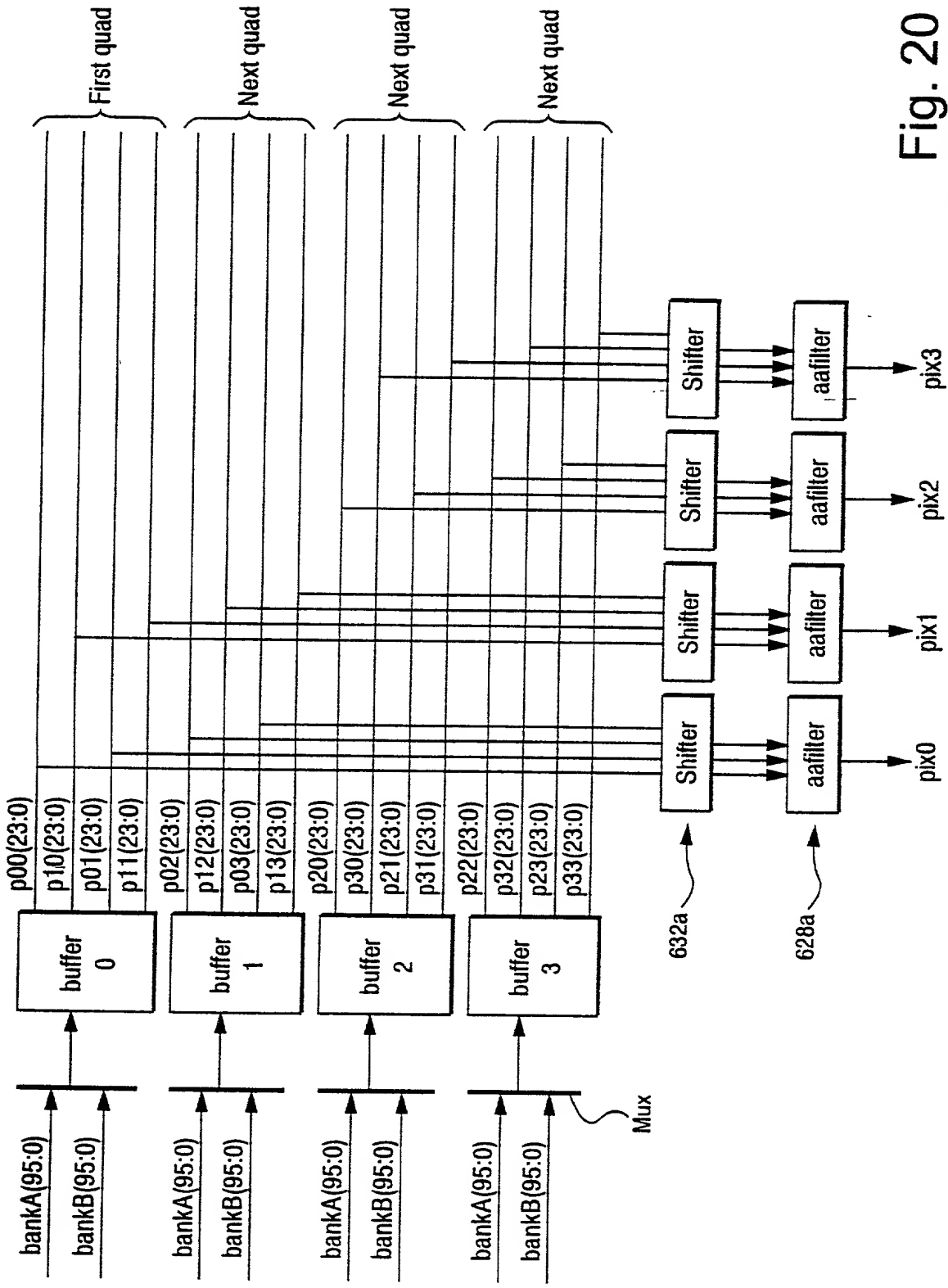
Fig. 20
(De-flicker buffering)

First quad

Next quad

Next quad

Next quad

p00(23:0)
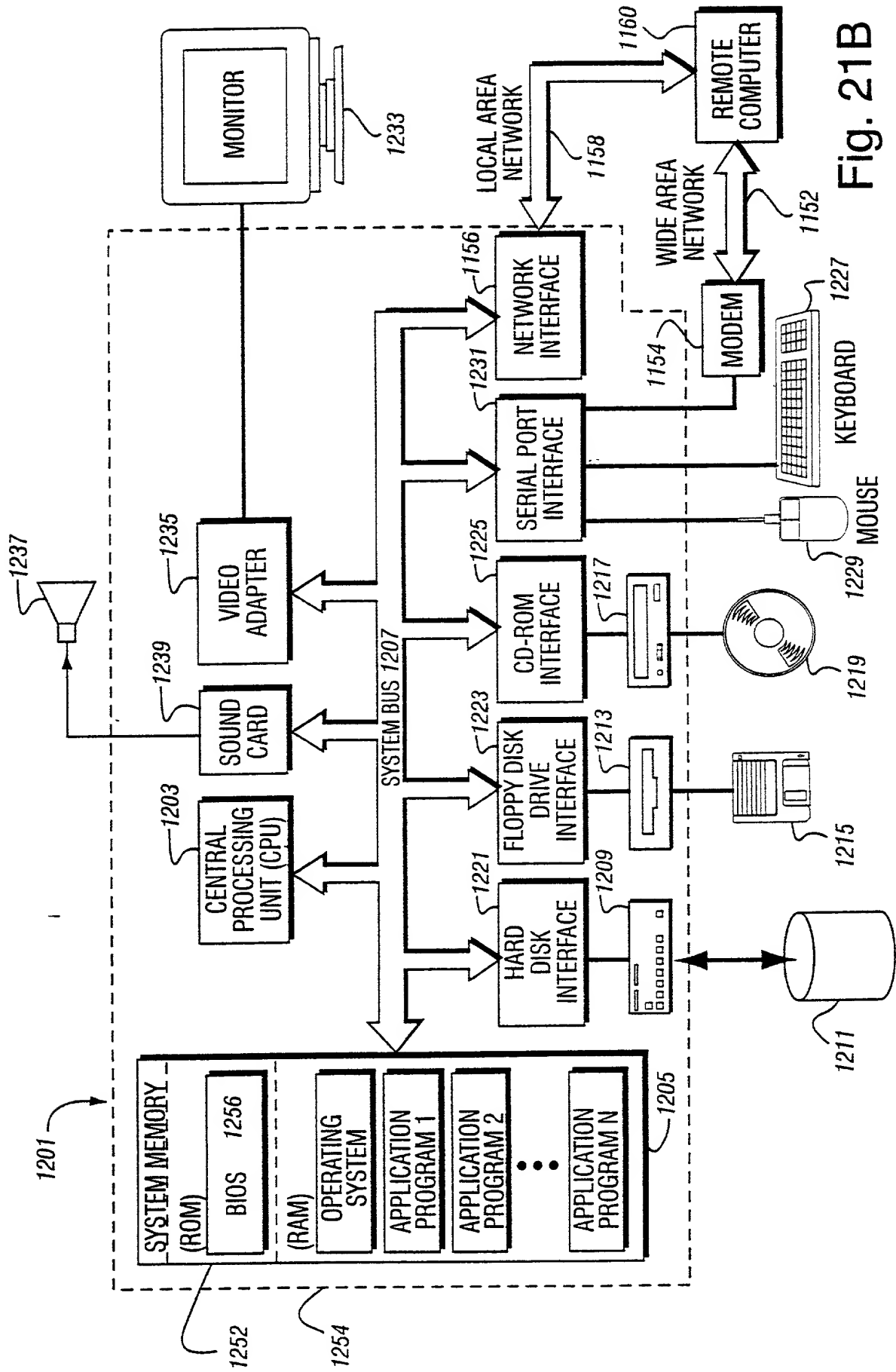p10(23:0)
p01(23:0)
p11(23:0)

p02(23:0)
p12(23:0)
p03(23:0)
p13(23:0)

p20(23:0)
p30(23:0)
p21(23:0)
p31(23:0)

p22(23:0)
p32(23:0)
p23(23:0)
p33(23:0)

buffer 0

buffer 1

buffer 2

buffer 3

bankA(95:0)
bankB(95:0)

bankA(95:0)
bankB(95:0)

bankA(95:0)
bankB(95:0)

bankA(95:0)
bankB(95:0)

Mux

Shifter

Shifter

Shifter

Shifter

aafilter

aafilter

aafilter

aafilter

pix0

pix1

pix2

pix3

632a

628a

Fig. 21A

Fig. 21B